University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Report

Keyword Spotting with an Arduino Nano 33 BLE Sense

Author:	Sadegh Naderi
Matriculation No.:	7024414
Author:	Achal Shakywar
Matriculation No.:	7025278
Author:	Malik Al Ashtar Ghansletwala
Matriculation No.:	7025306
Course of Studies:	Masters in Business Intelligence and Data Analytics
First examiner:	Prof. Dr. Elmar Wings
Submission date:	February 12, 2024

University of Applied Sciences Emden/Leer · Faculty of Technology · Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

API Application Programming Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

GPU Graphics Processing Unit

IMU Inertial Measurement Unit

IoT Internet of Things

KDD Knowledge Discovery in Databases

Part I.

Introduction

1. Introduction

Voice recognition has become a ubiquitous feature in our modern lives, powering everything from virtual assistants to smart home devices [Waq+21]. However, most voice recognition systems rely on cloud-based processing, which raises concerns about data privacy, security, and the need for a consistent internet connection [DB21; Gim+22]. The "Keyword Spotting with Arduino Nano 33 BLE Sense" project sets out to address these challenges by bringing voice recognition to the edge. The TinyML paradigm in embedded machine learning seeks to transfer the abundance of processing tasks from conventional high-end systems to lower-end client devices [Ray22]. This project embodies the fusion of machine learning and embedded systems, showcasing the potential for keyword spotting on a compact, low-power, and feature-rich device.

The primary objective of this project is to enable the Arduino Nano 33 BLE Sense to recognize specific keywords or phrases directly on the device. By doing so, we demonstrate the feasibility of edge-based voice recognition, reducing the need for cloud connectivity and enhancing user privacy. This journey encompasses data collection, audio preprocessing, machine learning model training, and the deployment of a compact and energy-efficient model capable of detecting predefined keywords from real-time audio input. The Arduino Nano 33 BLE Sense, known for its compact form factor and resource efficiency, serves as a compelling platform to illustrate the potential of machine learning at the edge.

This project represents a tangible step towards creating intelligent, privacy-conscious, and offline-capable voice-activated solutions that can be applied across various domains, from home automation to assistive technologies. Join us as we embark on an exciting exploration at the cutting edge of technology, delving into the world of Keyword Spotting with the Arduino Nano 33 BLE Sense, and paving the way for more versatile, privacy-aware interactions with voice-activated technology

1.1. Challenges

- **Limited computational resources and memory:** Adapting machine learning models to run efficiently on the Arduino Nano 33 BLE Sense's constrained computational resources is a central challenge. You'll need to optimize and quantize your model to ensure

it works well within the device's limitations, striking a balance between model complexity and accuracy. The small size of SRAM and flash memory in edge devices, often less than 1 MB, poses a challenge for deploying machine learning tasks at the edge [Ray22].

- **Noise and variability handling:** Designing a robust keyword spotting system that can handle variations in speech, accents, and environmental noise is essential. Building a model that is resilient to different conditions and speaking styles can be challenging, and it's crucial for practical applications.
- **Hardware and software heterogeneity:** Variability in hardware and software infrastructure complicates the adoption of consistent learning and deployment strategies for TinyML systems [Ray22].
- **Lack of suitable datasets:** Existing datasets may not align with TinyML architecture requirements, emphasizing the need for standardized datasets tailored for low-power edge devices [Ray22].

1.2. Proposed Solution

To tackle limited computational resources, optimize the machine learning model, use lightweight architectures, and explore hardware acceleration options. Achieving real-time processing involves streamlining code, leveraging multithreading, and potentially reducing the audio sampling rate. To address noise and variability, augment the training data, implement noise reduction techniques, and consider adaptive models for improved robustness to real-world audio variations.

1.3. Structure

The report begins with an introduction that outlines the challenges, proposed solution, and report structure. In the next chapter, the focus shifts to Domain Knowledge, covering machine learning deployment for embedded systems, including topics like TinyML, data collection, preprocessing, model training, deployment, and various hardware components and tests. The following chapter provides a detailed Hardware Description, focusing on the Arduino Nano 33 BLE Sense and its sensors. The subsequent chapter explores Software Description, encompassing tools like Arduino IDE, TinyML model development, audio processing libraries, and machine learning frameworks. The report then delves into Data Mining, specifically Convolutional Neural Network (CNN) concepts and applications. Important Python packages, including TensorFlow and NumPy, are discussed in the subsequent chapter.

The Knowledge Discovery in Databases (KDD) Process is introduced and detailed in the next chapter, followed by a chapter on Data Transformation and Data Mining in KDD. The deployment methods, including manual deployment and TensorFlow Lite, are covered in the next chapter. The Requirements, including a Bill of Materials and Development Environment details, are discussed in the subsequent chapter. The Program Flowchart is presented, followed by a chapter on Documentation Development. Software Tests are detailed, emphasizing automation with pytest. Results are discussed, covering data transformation, model export, training, and Arduino Nano 33 BLE Sense results. Finally, the report concludes with a summary, outlining future work and tasks to be addressed.

1.3.1. What it is used for

The board is programmed to recognize keywords "yes" and "no," showing green and red LED light colors in response. As the majority of the spoken input is not relevant to the voice interface [War18], a blue LED light is assigned for unknown keyword. The device can become portable by using a powerbank. Nevertheless, it is important to highlight that the primary purpose of this project is for educational use.

It is worthwhile to mention that the task of keyword spotting is distinct from the type of speech recognition performed on a server once an interaction has been identified [War18]:

- The majority of their input comprises silence or background noise rather than speech.
- As was mentioned, most of the speech input is unrelated to the voice interface, necessitating models that are unlikely to trigger on arbitrary speech.
- Recognition in keyword spotting focuses on individual words or short phrases, not entire sentences.

1.3.2. First Steps: Requirements and Installation

The project

The first crucial steps involve installing the necessary software components. Begin by installing Python on your system, and you can obtain the latest version from the official Python website. If you're using Linux, the installation process can be initiated with commands like `sudo apt-get update` and `sudo apt-get install python3`.

Following this, proceed to download and install PyCharm, a widely used Python IDE, from the JetBrains website. Once both Python and PyCharm

are in place, the next step is to create and set up a virtual environment for your project. Execute commands such as `python -m venv venv` to create the virtual environment and `venv\Scripts\activate` to activate it. Subsequently, install dependencies listed in the `requirements.txt` file, including any specific developer packages required for your project. See sections 12.3 and 15.2 and Chapter 13 for more details.

In addition, download and install the Arduino IDE from the official Arduino website. This is a fundamental platform for Arduino development and serves as the environment for coding, compiling, and uploading code to Arduino boards. Secondly, install the TensorFlow Library for Arduino IDE by referring to the instructions outlined in the TensorFlow Lite for Microcontrollers documentation, accessible at the provided link: TensorFlow Lite Micro.

Additionally, for Windows users, it is essential to set up environment variables to work seamlessly with TensorFlow and Arduino in the Command Prompt.

To start with the code, visit the directory `Code/KeywordSpotting`.

See Section 15.2 for more information about the setup steps.

The device

Start by connecting the device to your computer using a USB cable. Ensure that the Arduino Nano 33 BLE Sense is recognized by your computer, and, if necessary, upload your desired code using the Arduino IDE or another development environment.

You can proceed to power on the Arduino Nano 33 BLE Sense independently using a power bank. Connect the power bank to the Arduino Nano 33 BLE Sense using a USB cable, turn on the power bank, and verify that the board is operating as expected.

See chapter 11 for more information.

1.3.3. Working with Examples

To work with the TensorFlow Lite Micro library for Arduino, follow these steps for installation and usage:

Installation

Clone the tflite-micro-arduino-examples GitHub repository into the libraries folder of the Arduino IDE. The installation location varies by operating system but is typically in `~/Arduino/libraries` on Linux, `~/Documents/Arduino/libraries/` on MacOS, and `My Documents\Arduino\Libraries` on Windows.

Use the following commands in the terminal:

```
git clone https://github.com/tensorflow/tflite-micro-arduino-
examples Arduino_TensorFlowLite
cd Arduino_TensorFlowLite
```

To update the repository, use:

```
cd Arduino_TensorFlowLite
git pull
```

Usage

After installation, launch the Arduino IDE, and you'll find an `Arduino_TensorFlowLite` entry in the `File -> Examples` menu. This submenu provides sample projects for experimentation, such as the "Hello World" example.

The library is designed for the Arduino Nano 33 BLE Sense board, but the framework code for running machine learning models is compatible with most Arm Cortex M-based boards. Note that specific code for accessing peripherals like microphones, cameras, and accelerometers is tailored to the Nano 33 BLE Sense.

Part II.

Domain Knowledge

2. Foundations and Concepts in Machine Learning Deployment for Embedded Systems

The implementation of TinyML for training datasets on the Arduino Nano BLE Sense 33 to activate the RGB LED in response to valid voice commands involves several critical aspects:

2.1. TinyML

Tiny Machine Learning (TinyML) is a machine learning technique that integrates reduced and optimized machine learning applications that require "full-stack" (hardware, system, software, and applications) solutions. It includes machine learning architectures, techniques, tools, and approaches capable of performing on-device analytics at the very edge of the cloud. TinyML can be implemented in low-energy systems, such as sensors or microcontrollers to perform automated tasks. The technique is still ML, but with less energy and costs and without an internet connection [Rib 1].

TinyML introduces Machine Learning to the scene by incorporating Artificial Intelligence into small hardware components. Tiny machine learning is a rapidly growing field of machine learning technologies and applications that includes hardware (dedicated integrated circuits), algorithms, and software capable of performing on-device sensor data analytics at extremely low power.

In summary, the application of TinyML for training datasets on the Arduino Nano BLE Sense 33 to blink its RGB LED in response to valid voice commands involves a combination of hardware, software, and machine learning techniques, tailored to the device's constraints and requirements. This technology enables voice-controlled interactions in embedded systems, making it suitable for a variety of applications, from home automation to assistive devices[Mis 1].

2.2. Data Collection

To train a TinyML model for voice command recognition, a dataset of audio samples containing both the target keywords and background noise should be collected. These samples are recorded using the onboard microphone of the Arduino Nano BLE Sense 33.

2.3. Data Preprocessing

Data preprocessing involves converting the raw audio data into a suitable format for model training. This may include feature extraction, such as MFCC (Mel-frequency cepstral coefficients), to represent the audio in a way that is amenable to machine learning. Please refer to Figure 2.1

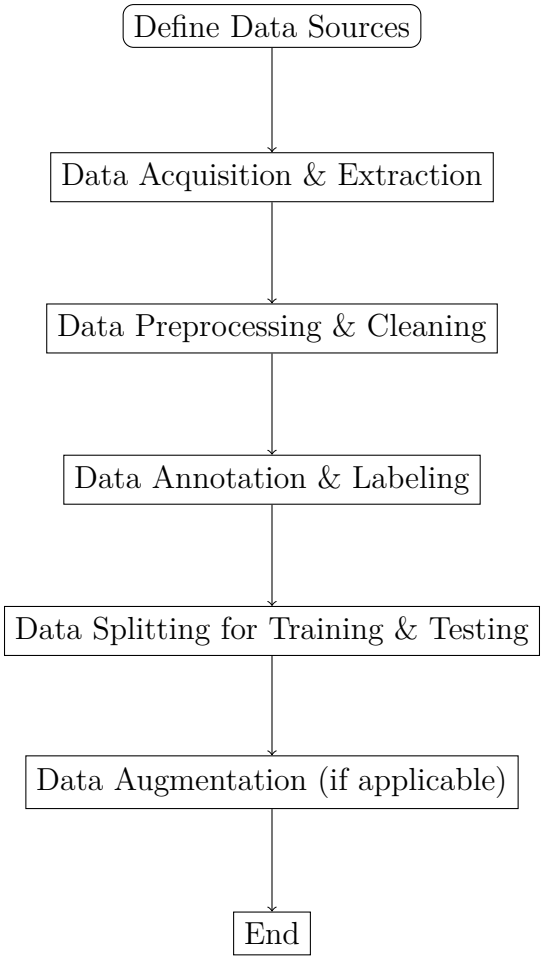


Figure 2.1.: Flow of Data Preprocessing

2.4. Model Training

The collected and preprocessed audio data is used to train a TinyML model. Training may involve techniques like transfer learning, quantization, and model compression to ensure the model's size and computational requirements are compatible with the Arduino Nano BLE Sense 33's limitations. See Figure 2.2

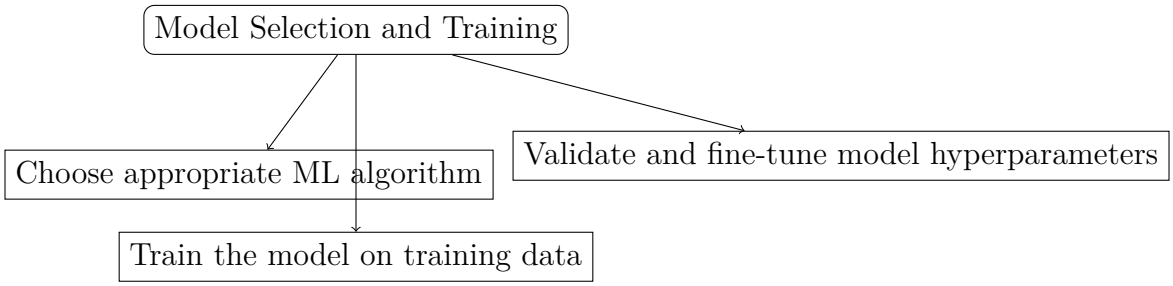


Figure 2.2.: Flow of Model Training

2.5. Model Deployment

Once the TinyML model is trained, it is deployed onto the Arduino Nano BLE Sense 33. The model must be integrated into the device's firmware and take advantage of the available software libraries for inference. See Figure 2.3

2.6. Keyword Recognition

In the deployed system, the TinyML model continuously analyzes audio data from the onboard microphone to recognize specific keywords or commands. When a valid keyword is detected, a response is triggered to blink the RGB LED on the Arduino Nano BLE Sense 33.

2.7. Feedback and Optimization

Continuous monitoring and feedback are essential for improving the recognition performance. Optimization may include fine-tuning the model, enhancing noise filtering, and refining the response mechanism to ensure accurate and reliable keyword recognition.

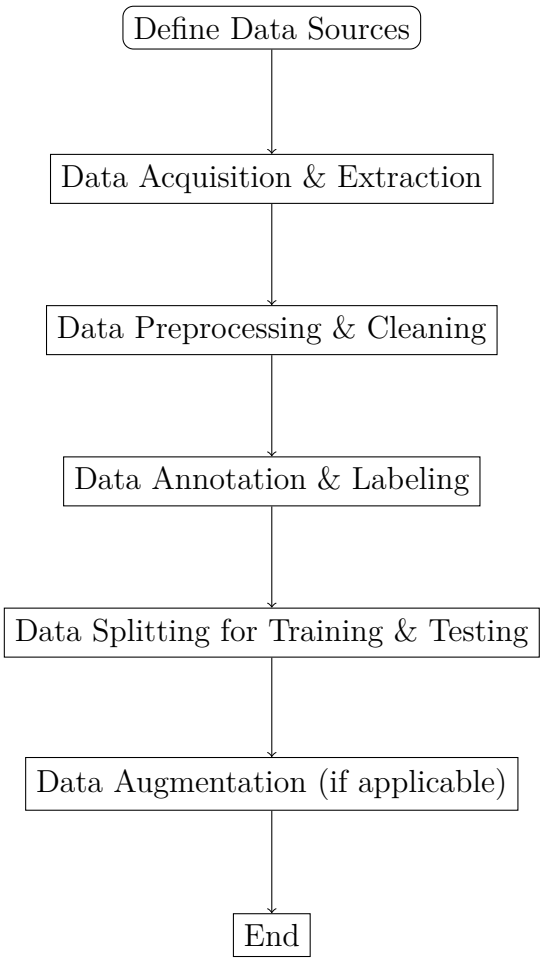


Figure 2.3.: Block Diagram for Model Deployment

2.8. Power Management

Since the Arduino Nano BLE Sense 33 runs on a limited power source, power management is a critical consideration. Implementing low-power modes and strategies to minimize energy consumption is essential for prolonged operation.

2.9. Outliers

In Machine Learning, an **outlier** is a data point that deviates significantly from the rest of the dataset. They are often abnormal observations that skew the data distribution, and arise due to inconsistent data entry, or erroneous observations. Outliers can skew and mislead the training process of machine learning algorithms resulting in longer training times, less accurate models, and ultimately poorer results [Nic 0].

Outliers can be detected using various techniques such as:

1. **Standard Deviation:** When the data, or certain features in the dataset, follow a normal distribution, you can use the standard deviation of the data, or the equivalent z-score to detect outliers [C 0].

2. **Clustering based outlier detection:** In the K-Means clustering technique, each cluster has a mean value. Objects belong to the cluster whose mean value is closest to it. In order to identify the Outlier, firstly we need to initialize the threshold value such that any distance of any data point greater than it from its nearest cluster identifies it as an outlier for our purpose.

In the context of **speech recognition**, outliers can be particularly challenging. For example, a segment-based speech recognizer built on a neural net has to identify outliers as well. One approach is to artificially generate outlier samples, but this is tedious, error-prone, and significantly increases the training time. An alternative is applying a replicator neural net for this task, originally proposed for outlier modeling in data mining. This approach allows the recognizer to perform similarly without the need for a large amount of outlier data.

Please note that dealing with outliers often requires domain expertise, and none of the outlier detection techniques should be applied without understanding the data distribution and the use case.

2.10. Anomalies

In Machine Learning, **anomalies** are data points that stand out from other data points in the dataset and don't confirm the normal behavior in the data[Kum 1]. These data points or observations deviate from the dataset's normal behavioral patterns[Joh 1]. Anomaly detection is an unsupervised data processing technique to detect anomalies from the dataset.

Anomaly detection plays an instrumental role in robust distributed software systems. It can enhance communication around system behavior, improve root cause analysis, reduce threats to the software ecosystem, and is commonly used for data cleaning, intrusion detection, fraud detection, systems health monitoring, event detection in sensor networks, and ecosystem disturbances.

In the context of **speech recognition**, anomalous audio in speech recordings is often caused by speaker voice distortion, external noise, or even electrical interferences. These obstacles have become a serious problem in some fields, such as high-quality dubbing and speech processing. A novel approach using a temporal convolutional attention network (TCAN) is proposed to tackle this problem. The use of temporal conventional

network (TCN) can capture long-range patterns using a hierarchy of temporal convolutional filters. To enhance the ability to tackle audio anomalies in different acoustic conditions, an attention mechanism is used in TCN, where a self-attention block is added after each temporal convolutional layer[HH21]. This aims to highlight the target-related features and mitigate the interferences from irrelevant information.

3. Hardware Description

Arduino stands as an open-source electronics platform, boasting adaptable and user-friendly hardware and software. Featuring on-board microcontroller and microprocessor kits, it facilitates the creation of both digital and analog devices. The microcontroller, often referred to as a miniature computer embedded on the Arduino board, typically requires additional electronic components such as diodes, resistors, capacitors, and transistors to manage voltage and current. However, the Arduino team has ingeniously designed a user-friendly environment to streamline hardware operation through software. Simply power the board to the required voltage, script the desired program, and upload it within seconds. This approach eliminates the need to grapple with intricate electronic configurations, granting independence from concerns about hardware intricacies.

As technological advancements in the semiconductor and electronics industry unfold, control problems are increasingly addressed using these compact microcontrollers instead of traditional mechanical and electrical switches. Across all Arduino boards, a common denominator is the microcontroller, essentially a diminutive computer pivotal in facilitating edge computing applications.[Ard12]

3.1. Arduino Nano 33 BLE Sense

The Arduino Nano Family encompasses a series of boards characterized by a compact footprint yet rich in features. This lineup spans from the economical entry-level Nano to the advanced Nano BLE Sense/Nano RP2040 Connect, incorporating Bluetooth/Wi-Fi radio modules. Additionally, these boards are equipped with an array of built-in sensors, including temperature, humidity, pressure, gesture, microphone, and more. Notably, they are programmable with MicroPython and are compatible with machine learning applications.[Raj19]

The Arduino Nano 33 BLE Sense operates at a maximum of 3.3V, ensuring that its digital and analog pins are never subjected to higher voltages. Additionally, its Bluetooth Low Energy (BLE) module enhances its suitability for IoT applications. The board houses an nRF52840 processor with a clock speed of 64 megahertz, 256 kilobytes of Static RAM (SRAM), and 1 megabyte of flash memory. With 14 digital I/O pins, including eight analog input pins for external components and sensors,

the board draws an impressively low 10 mA current per I/O pin.

Dimensions: *The Arduino Nano 33 BLE Sense has precise dimensions of 45mm in lenght, 18mm in width, and its thickness is around 7mm.*

The nomenclature of the Arduino Nano 33 BLE Sense model itself conveys essential details. The designation "Nano" reflects its diminutive size, categorized as nano. Furthermore, the term "BLE" denotes support for Bluetooth Low Energy, while "Sense" indicates the incorporation of on-board sensors. These sensors encompass an accelerometer, gyroscope, magnetometer, temperature and humidity sensor, pressure sensor, proximity sensor, color sensor, gesture sensor, and even an integrated microphone.[Raj19]

To enable RGB color analysis and person detection, it's essential to establish a connection between the BLE 33 Sense interface and the Arducam OV2640 camera shield. The Arducam OV2640 camera is employed for tasks such as RGB detection, object detection, and gesture recognition. The combination of the Arduino Nano 33 BLE Sense and the Arducam camera shield proves to be an ideal pairing for the development of machine learning (ML) and artificial intelligence (AI) applications. With a comprehensive set of on-board sensors, leveraging this setup involves the straightforward installation of relevant libraries on the Arduino board, seamlessly supporting the functionality of sensors and facilitating the implementation of machine learning applications.

The following figure shows top view and bottom view of Arduino Nano 33 BLE Sense,

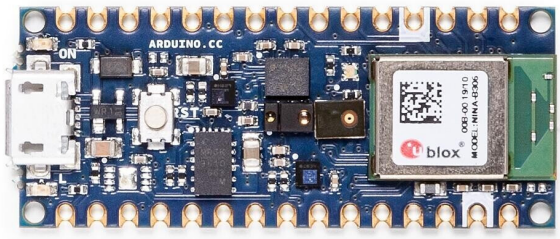


Figure 3.1.: Top view of Arduino Nano 33 BLE Sense

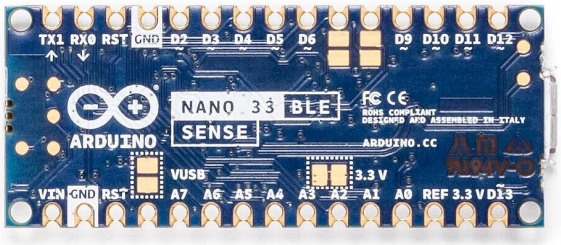


Figure 3.2.: Bottom view of Arduino Nano 33 BLE Sense

The Nano board with Bluetooth Low Energy (BLE) capabilities is both compact and reliable, incorporating the NINA B306 module for BLE and Bluetooth 5 communication. This module is built on the Nordic nRF52480 processor, housing a robust Cortex M4F Central Processing Unit (CPU). The architecture seamlessly aligns with both Arduino IDE Online and Offline. The Arduino Nano BLE 33 Sense is equipped with a set of on-board sensors, namely ADPS-9960, LPS22HB, HTS221, LSM9DS1, and MP34DT05-A. Despite its small size, this board encompasses all the necessary sensors on board. The Arduino Nano 33 BLE Sense have the following set of Sensors, BLE module and its functionality below.[Ard12]

- The Bluetooth is managed by a NINA B306 module.
- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor.
- The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.
- The LPS22HB reads barometric pressure and environmental temperature.
- The MP34DT05 is support the sound detection.
- The HTS221 senses relative humidity.

3.2. On-Board Sensor Description

The Arduino Nano 33 BLE Sense is equipped with a range of embedded sensors on the board, commonly employed for measuring both analog and digital values in the surrounding environment. With compact dimensions of 45mm $\times$ 18mm, this board proves highly valuable for Internet of Things (IoT) and Artificial Intelligence (AI) applications, serving as an embedded device where space is a critical constraint. Operating at a low power consumption of 3.3V, this small-sized board can efficiently function on small batteries for extended periods, making it suitable for various applications.[Ard12] Thanks to its onboard sensors, minimal power requirements, and compact architecture, the Nano board finds versatility in deployment. The Arduino Nano 33 BLE Sense introduces a new board on a familiar form factor. For in-depth information about each component and datasheets for individual sensors, detailed information can be accessed through the provided links. Brief descriptions of each sensor are outlined below.

- The sensor MP34DT05 is the digital microphone. it is useful for capturing, analyzing and detecting the sound in real time.
- The sensor HTS221 senses the relative humidity, and temperature, to get highly accurate measurements of the environmental conditions.
- The sensor LPS22HB is a barometric pressure sensor, it measures the environmental pressure which is usefull for simple weather station monitoring
- The sensor LSM9DS1 is a 9 axis Inertial Measurement Unit (IMU) use as a accelerometer, gyroscope, and magnatometer, this 9 axis sensor is ideal for wearable devices.
- The USB port allows you to connect Arduino Nano 33 BLE sense to your machine.
- The ADPS-9960 is a digital proximity, ambient light, RGB and gesture sensor. it can measure the proximity distance, light, color and gestures when moving close with the borad.
- There are 3 different LEDs that can be accessed on the Nano BLE Sense: RGB Programmable LED , the built-in orange Programmable LED and the Power LED.

Figure below illustrates the onboard embedded sensors, featuring a potent processor, the nRF52840 from Nordic Semiconductors. This processor stands out among other Arduino boards, boasting a 32-bit ARM® Cortex™-M4 CPU running at 64 MHz. It's crucial to consider the accuracy of sensor readings, which heavily depends on the placement of the sensors within the environment. This aspect holds significant importance.

Furthermore, it's essential to note the operational conditions. The operating temperature should not surpass 85°C and should not fall below -40°C. Additionally, factors such as humidity levels and air pressure values should be carefully monitored.

3.2.1. Gesture, Proximity, and Color Detection Sensor ADPS-9960

The APDS-9960 device incorporates advanced features including Gesture detection, Proximity detection, Digital Ambient Light Sense (ALS), and Color Sense (RGB). Gesture detection employs four directional photodiodes to detect reflected infrared (IR) energy, emitted by the integrated LED. This process transforms physical motion information such as velocity, direction, and distance into digital data.[Ard12]

For the following Applications the sensor is in use:

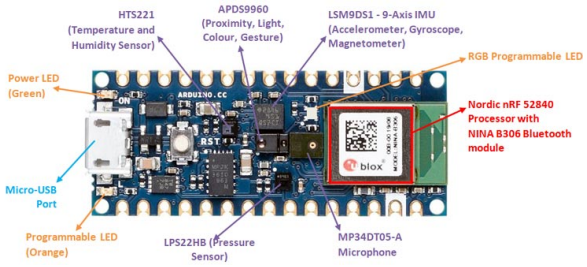


Figure 3.3.: : Components in Arduino Nano 33 BLE Sense

Table 3.1.: Technical Specifications of Arduino Nano 33 BLE Sense

Feature	**Specification**
Microcontroller	nRF52840
Operating Voltage	3.3V
Input Voltage (Limit)	21V
DC Current per I/O Pin	15 mA
Clock Speed	64MHz
CPU Flash Memory	1MB
SRAM	256KB
LED_BUILTIN	13
IMU (Accelerometer, Gyroscope, Magnetometer)	LSM9DS1
Microphone	MP34DT05
Gesture, Light, Proximity, Color Sensor	APDS9960
Barometric Pressure Sensor	LPS22HB
Temperature, Humidity Sensor	HTS221

- Gesture Detection
- Color Sense
- Ambient Light Sensing
- Proximity Sensing

3.2.2. Accelerometer, Gyroscope, and Magnetometre Sensor LSM9DS1

The LSM9DS1 is a system-in-package that integrates a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. Inertial Measurement Units (IMUs) operate by detecting rotational movements along three axes that is Pitch, Roll, and Yaw. This process relies on the collaboration of an Accelerometer, Gyroscope, and Magnetometer. The Accelerometer provides information about the velocity of the Inertial Measurement Unit (IMU) module's movement.

The Gyroscope measures the rate of rotational movement on the IMU. Additionally, the Magnetometer gauges the force of gravity acting on the IMU.

For the following Applications the IMU is in use:

- Sports Technology - helping athletes to know how they can improve their movements.
- Compact transportation solutions like Segway.
- Consumer electronics; Smartphones, tablets, fitness trackers for motion sensing and orientation.
- Display/map orientation and browsing.
- Gaming and virtual reality input devices.
- Advanced gesture recognition.
- Indoor navigation.

Utilizing an Inertial Measurement Unit (IMU) comes with certain drawbacks and considerations. The primary drawback is the presence of accumulated error, commonly referred to as 'Drift.' This occurs due to the constant measurement of changes and the rounding off of calculated values. Over an extended duration, this process can result in notable errors. To mitigate the impact of drift, it is advisable to employ a high-quality IMU sensor and ensure proper calibration of the sensor.[Lsm]

Calibration of an IMU

During the research, it was discovered that several methods exist for calibrating the involved sensors. The specific time intervals between each calibration are not explicitly defined; nevertheless, it is recommended to perform regular calibration, particularly when unusual outputs are observed[MB15]. Below, we provide a brief overview of some calibration methods:

Low and High Limit Method

In this approach, the sensor undergoes circular rotations along each axis multiple times. The midpoint is subsequently determined between the two extremes. If there is no offset, the midpoint aligns closely with zero. However, if there is a slight deviation from zero, this value represents the hard iron offset, stemming from the distortion induced by the Earth's magnetic field. Primarily, this method is employed for calibrating the Magnetometer.[MB15]

Magneto V1.2

In this technique, the raw magnetometer data undergoes pre-processing with axis-specific gain correction, transforming the raw output into nanoTesla.

```
Xm_nanoTesla = rawCompass.m.x*(100000.0/1100.0);
% Gain X [LSB/Gauss] for selected input field range
Ym_nanoTesla = rawCompass.m.y*(100000.0/1100.0);
Zm_nanoTesla = rawCompass.m.z*(100000.0/980.0);
```

This converted data is saved into the file `Mag_raw.txt` that you open with the Magneto program. To start using this method, we first need to replace the (100000.0/1100.0) scaling factors with values that convert your specific sensors output into nanoTesla. Rather than simply finding an offset and scale factor for each axis, Magneto creates twelve different calibration values that correct for a whole set of errors: bias, hard iron, scale factor, soft iron and misalignment.

An additional advantage of this method is its applicability for accelerometer calibration. Once again, you may need to preprocess the specific raw accelerometer output, considering factors such as bit depth and G sensitivity, to transform the data into milliGalileo. Subsequently, inputting a value of 1000 milliGalileo serves as the "norm" for the gravitational field.[MB15]

3.2.3. Pressure Sensor LPS22HB

The LPS22HB serves as an ultra-compact piezoresistive absolute pressure sensor, operating as a digital output barometer. It finds application in the following scenarios:

- Sports watches
- Altimeters and barometers for portable devices
- Weather station equipment

An installed sensor element, along with an IC interface, communicates through an I2C or SPI bus. Its functionality is specified within a temperature range from -40°C to +85°C.

- Absolutdruckbereich: 260 bis 1260 hPa
- 16-bit Temperaturdatenausgabe
- Versorgungsspannung: 1,7 bis 3,6 Volt
- 24-bit Druckdatenausgabe

3.2.4. Relative Humidity and Temperature Sensor HTS221

The HTS221 is a highly compact sensor designed for measuring relative humidity and temperature. It incorporates a sensing element and a mixed signal to deliver measurement information via digital serial interfaces. This sensor is employed in the following applications:

- Air conditioning, heating and ventilation
- Smart home automation
- Industrial automation
- Air humidifiers
- Refrigerators

This sensor communicates via the I2C and SPI buses. It is operational within a temperature range from -40°C to +120°C. Power supply in the range of 1.7 to 3.3 volts is required. Temperature measurement is conducted with an accuracy of ±5°C.

- SDA/SDI/SDO - I2C serial Data (SDA) and 3 wire-SPI serial data input/output (SDI/SDO)
- SCL/SPC - I2C serial clock (SCL) and SPI serial port clock (SPC)
- DRDY - Data ready output signal
- SPI enable - I2C/SPI mode selection
- VDD - Stromversorgung
- GND - Ground

3.2.5. Digital Microphone MP34DT05-A

The MP34DT05-A is a highly compact and energy-efficient digital microphone featuring omnidirectional capabilities. It incorporates a capacitive sensing element and an IC interface. The sensing element, designed for detecting acoustic waves, undergoes manufacturing through a specialized silicon micromachining process dedicated to audio sensor production. The MP34DT05 is characterized as a low-distortion microphone with a signal-to-noise ratio of 64 dB, a sensitivity of -26 dBFS ± 3 dB, and an Acoustic Overload Point (AOP) at 122.5 dBSPL. The microphone produces a PDM signal as its output, which is a binary signal modulated by Pulse Density Modulation (PDM) from the analog signal[Mp3]. The sensor is utilized in the following applications:

- Portable media player
- Mobile Terminal
- Speech recognition

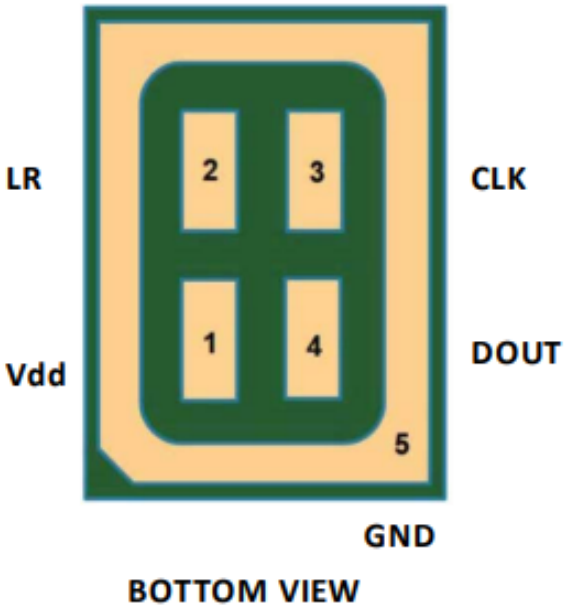


Figure 3.4.

Table 1. **Pin description**

Pin #	Pin name	Function
1	Vdd	Power supply
2	LR	Left/Right channel selection
3	CLK	Synchronization input clock
4	DOUT	Left/Right PDM data output
5 (ground ring)	GND	Ground

Figure 3.5.: Circuit diagram microphone

[Mp3]

3.2.6. Bluetooth Module nRF52840

The nRF52840 stands as an advanced and exceptionally versatile single-chip solution tailored to meet the growing demands of Ultra Low Power (ULP) wireless applications. This includes devices in close proximity, connected living environments, and the broader realm of the Internet

of Things (IoT) [Ard12]. Specifically designed to accommodate major feature advancements of Bluetooth 5, it leverages the enhanced performance capabilities introduced by Bluetooth 5.

Applications:

- Connected Health
- Wearables with wireless payment
- Advanced personal fitness devices
- Connected watches
- Smart city infrastructure
- Industrial mesh networks
- Smart Home products

3.3. Arduino Nano 33 BLE Pin Configuration

The Arduino Nano 33 BLE represents an advanced iteration of the Arduino Nano board, employing the robust nRF52840 processor. As illustrated in Figure 2.4, the board exhibits the subsequent pin configuration:

Digital Pins: There are 14 digital I/O pins that exclusively receive either HIGH or LOW values. These pins function as either input or output based on specific requirements. A HIGH state is achieved when the pins receive 5V, while a LOW state corresponds to a reception of 0V.

Analog Pins: The board boasts a total of 8 analog pins labeled A0 to A7. In contrast to digital pins, these analog pins can receive a range of values. They are instrumental in measuring analog voltages spanning from 0 to 5V.

PWM Pins: All digital pins on the board can be utilized as PWM pins, generating analog results through digital means.

SPI Pins: The board supports the serial peripheral interface (SPI) communication protocol, facilitating communication between the controller and peripheral devices such as shift registers and sensors. Two pins, Master Input Slave Output (MISO) and Master Output Slave Input (MOSI), are dedicated to SPI communication, facilitating the exchange of data.

I2C Pins: The board is equipped with the I2C communication protocol, a two-wire interface with SDL and SCL pins.

UART Pins: Featuring the UART communication protocol for serial communication, the board includes Rx and Tx pins. Rx serves as the receiving pin for serial data, while Tx functions as the transmission pin.

External Interrupts Pins: All digital pins can function as external interrupts, allowing the interruption of the main program with specific instructions in emergency scenarios.

LED at Pin 13 and AREF Pin: Pin 13 hosts an LED on the board, and AREF functions as a reference voltage for input voltage.

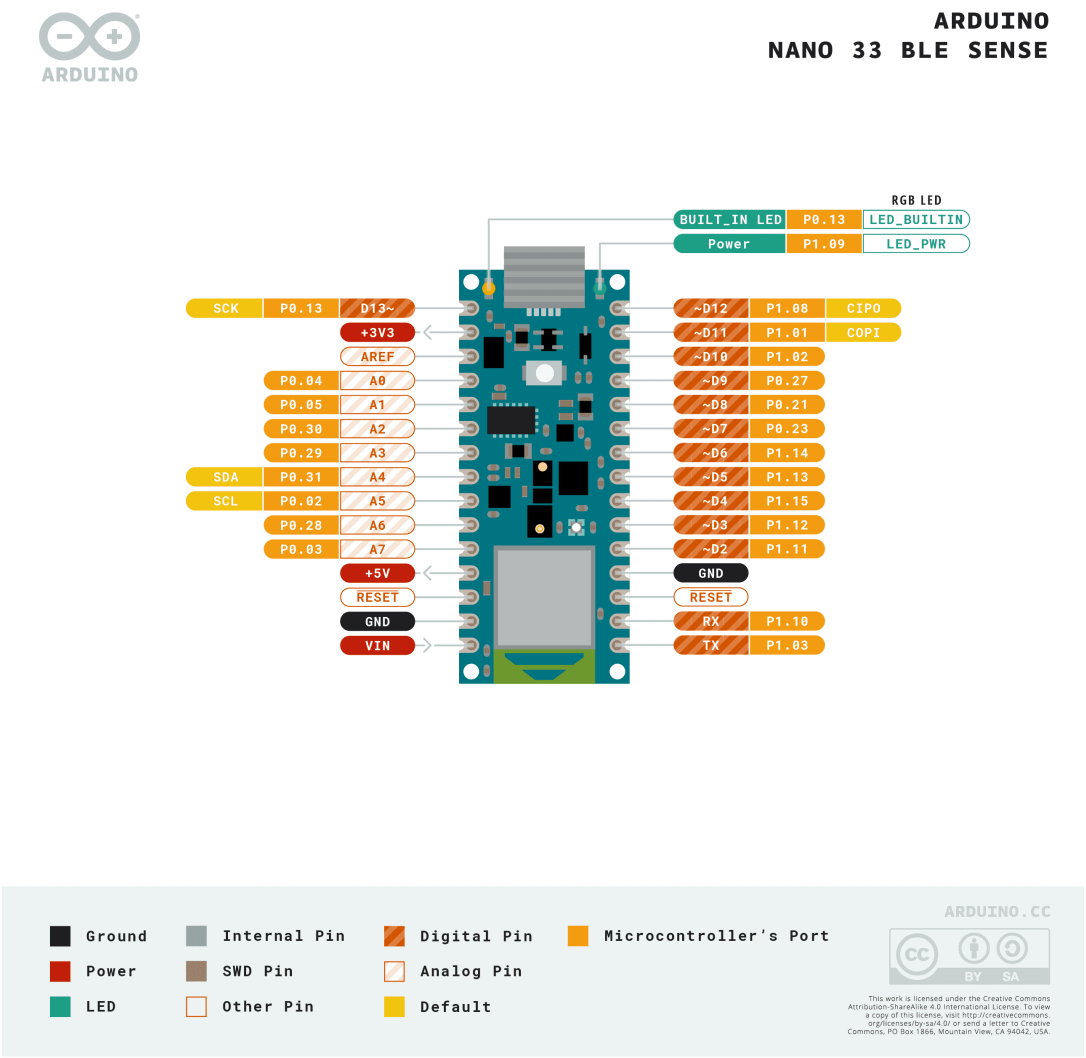


Figure 3.6.: Arduino Nano 33 BLE Pin Configuration

3.4. Hardware Tests

All the Arduino boards need power to operate, either it comes from the USB connection with Laptop, Ac power adapter, Battery or a regulated power supply. The most easiest way to operate arduino board is USB connection with laptop, normally these boards need 5V direct current

(DC) to operate. Arduino Nano 33 BLE sense also need these types of power sources for functionality, when applying one of the above mention power source the green LED glows as shown in the figure 3.7, it shows the sign of Arduino board working.

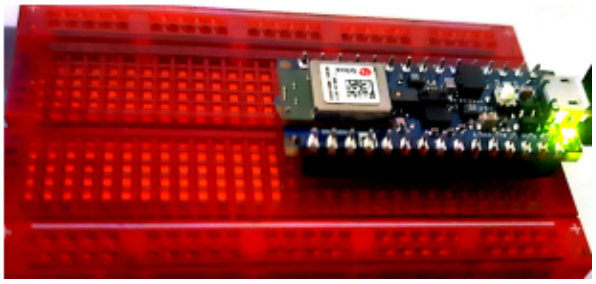


Figure 3.7.: Power On Arduino Nano 33 BLE Sense

3.4.1. Hardware parts

The Arduino Nano 33 BLE Sense is equipped with a variety of hardware features that make it particularly suitable for projects involving keyword spotting. Here's a breakdown of its relevant parts and it's functions and how they support the development and deployment of a keyword spotting application:

1. **nRF52840 Microcontroller (MCU)**
   - **Core Function:** This powerful ARM Cortex-M4 CPU runs at up to 64 MHz and is the brain of the Arduino Nano 33 BLE Sense. It's responsible for executing the code that processes input data, runs the machine learning (ML) model, and controls the output based on the model's inference.
   - **Keyword Spotting Relevance:** The MCU performs the real-time signal processing and inference tasks required for keyword spotting. It has enough computational power to run lightweight TinyML models directly on the device.
2. **Built-in LED:**
   - **Core Function:** The Arduino Nano 33 BLE Sense has a built-in RGB LED (Light Emitting Diode).
   - **Keyword Spotting Relevance:** The LED can be used to provide visual feedback based on keyword detection. For our project, it can light up in different colors or patterns to indicate the recognition of specific keywords.
3. **BLE (Bluetooth Low Energy) Capability:**

- **Core Function:** Allows wireless communication over Bluetooth.
- **Keyword Spotting Relevance:** This feature can be used to transmit the results of keyword spotting to other devices, such as smartphones, tablets, or computers. It enables the Arduino Nano 33 BLE Sense to function as part of a larger ecosystem, where it can send notifications or control other devices based on voice commands.

4. **Digital Microphone (MP34DT05):**
   - **Core Function:** This microphone captures audio signals, which are essential for any voice or sound-based application.
   - **Keyword Spotting Relevance:** It captures the user's keywords in the form voice commands. The audio data from the microphone is then processed and fed into the ML model for keyword detection.

3.4.2. Hardware Functions

Test with MP34DT05 (Digital Microphone)

Testing the MP34DT05 digital microphone on the Arduino Nano 33 BLE Sense for keyword spotting involves setting up the microphone, capturing audio data, and potentially sending the data to a keyword spotting model for analysis. The embed on-board MP34DT05 sensor in Arduino Nano 33 BLE Sense has the funcnality to sense audio voice from the environment. There is build in Arduino library for this particular sensor, which is PDM as shown in the figure.

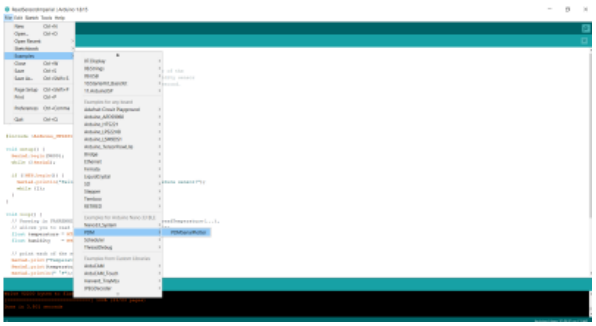


Figure 3.8.: MP34DT05, Digital Microphone

```
// RGB LED pins on the Arduino Nano 33 BLE Sense
const int RED_LED = 22;
const int GREEN_LED = 23;
const int BLUE_LED = 24;
```

Code/testhardware/microphone.c

Listing 3.1.: RGB LED pins on the Arduino Nano 33 BLE Sense

```
// Function to blink the LED with the specified color
void blinkLED(int red, int green, int blue) {
  digitalWrite(RED_LED, red);
  digitalWrite(GREEN_LED, green);
  digitalWrite(BLUE_LED, blue);
  delay(500);
  digitalWrite(RED_LED, LOW);
  digitalWrite(GREEN_LED, LOW);
  digitalWrite(BLUE_LED, LOW);
  delay(500);
}
```

Code/testhardware/microphone.c

Listing 3.2.: Function to blink the LED with the specified color

RGB LED pins on the Arduino Nano 33 BLE Sense

Function to blink the LED with the specified color

Initialization

Classification Results

- **Setup Function:**

  Initializes serial communication, configures the RGB LED pins as outputs, and initializes the Edge Impulse model with `Keywordspotting.begin()`. It checks for successful initialization of the model, halting the program if it fails.

- **Loop Function**
  - Runs the keyword spotting classifier with `Keywordspotting.run classifier(and result, false)` and checks the classifier's output.
  - Depending on the classification result, it blinks the LED in green, red, or blue. The decision is made based on the confidence level of the classification for "yes" and "no". If neither confidence level is above 0.8, it defaults to blinking blue, indicating an unrecognized keyword.

```
void setup() {
  // Initialize serial communication
  Serial.begin(115200);

  // Initialize the RGB LED pins
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(BLUE_LED, OUTPUT);

  // Initialize Edge Impulse model
  if (!Keywordspotting.begin()) {
    Serial.println("Failed to initialize Edge Impulse model")
    ;
    while (1);
  }
}

void loop() {
  // Run the machine learning model
  ei_impulse_result_t result = { 0 };
  if (Keywordspotting.run_classifier(&result, false) !=
   EI_IMPULSE_OK) {
    Serial.println("Failed to run classifier");
    return;
  }
```

Code/testhardware/microphone.c

Listing 3.3.: Initialization

```
  // Check the classification results and blink the LED
   accordingly
  if (result.classification[0].value > 0.8) { // Assuming "
   yes" is at index 0
    blinkLED(LOW, HIGH, LOW); // Blink green for "yes"
  } else if (result.classification[1].value > 0.8) { //
   Assuming "no" is at index 1
    blinkLED(HIGH, LOW, LOW); // Blink red for "no"
  } else {
    blinkLED(LOW, LOW, HIGH); // Blink blue for unrecognized
   words
  }
}
```

Code/testhardware/microphone.c

Listing 3.4.: Classification Results

- **Blink Function:**
  - `blinkLED(int red, int green, int blue)` controls the

RGB LED based on the specified color parameters. It turns on the LED with the color combination provided for half a second, then turns it off for half a second.

- **Considerations and Enhancements:**
  - **Model Indexing:** The code assumes that "yes" is at index 0 and "no" is at index 1 of the classification results. Ensure this aligns with how your model was trained and how the labels were indexed during the training process.
  - **Model Initialization and Error Handling:** The `Keywordspotting.begin()` and classifier run check for initialization and execution errors, which is good practice for detecting and handling runtime issues.

3.4.3. Responses of the Device

- As mentioned in the code, the `Green LED` blinks whenever the keyword `YES` is produced in form of voice command.

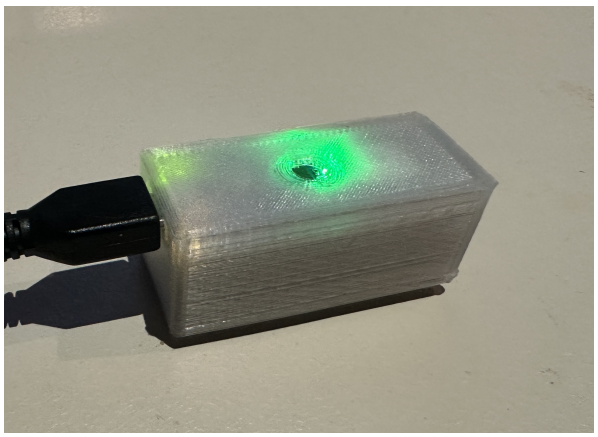


Figure 3.9.: Response to the keyword YES

- As mentioned in the code, the `RED LED` blinks whenever the keyword `NO` is produced in form of voice command.
- As mentioned in the code, the `BLUE LED` blinks whenever `UNRECOGNIZED KEYWORD` is produced in form of voice command.

3.4.4. Reset of Arduino

When developing and testing your keyword spotting model, we have likely made frequent adjustments to our code. The reset button allows for a quick way to restart the program to test new changes without the need to

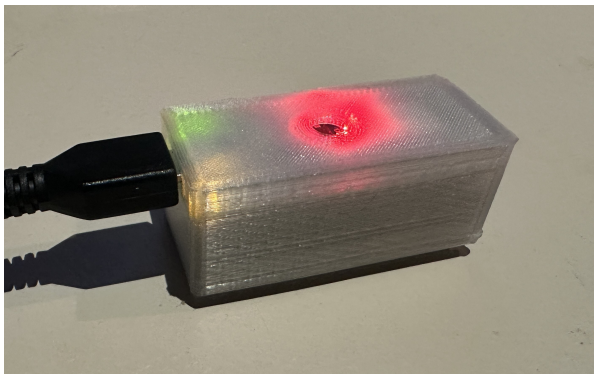


Figure 3.10.: Response to the keyword NO

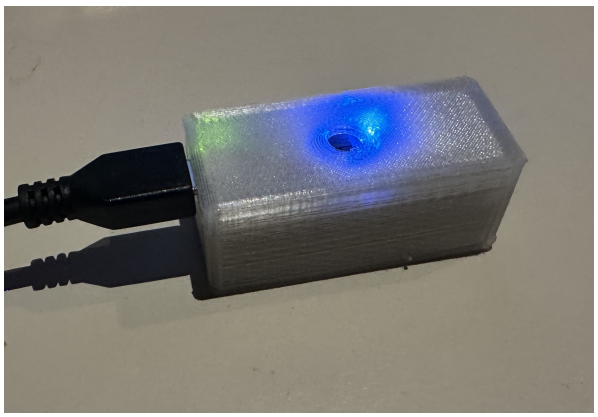


Figure 3.11.: Response to Unrecognized Keyword

disconnect and reconnect the board or use the software reset command. This is especially needed for rapid prototyping and debugging.

If our keyword spotting application enters an unexpected state or becomes unresponsive due to a bug or logic error, the reset button can quickly reboot the device, allowing you to resume testing without significant downtime.

Keyword spotting model and the associated signal processing is memory-intensive. If there are any memory leaks or if the program consumes increasing amounts of RAM over time, a reset can clear the memory, helping us to identify and troubleshoot memory-related issues.

3.4.5. Hardware Test Documentation

Overview

This document is designed to guide users through the process of verifying the functionality of the Arduino Nano 33 BLE Sense's digital microphone and built-in LED. These components are essential for developing a keyword spotting system that responds to voice commands with visual feedback.

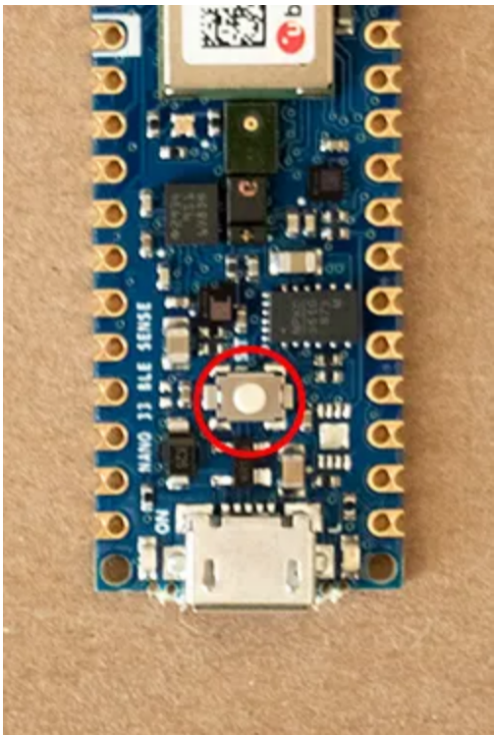


Figure 3.12.: Reset Button of Arduino

Test Environment Setup

Equipment Required

- Arduino Nano 33 BLE Sense board
- Computer with Arduino IDE installed
- USB cable for connection
- Sound source for testing the microphone
- A well-lit area to observe LED brightness and color accurately

Software and Libraries

- Arduino IDE
- PDM library for processing digital microphone input
- Basic LED control sketches

Preparation Steps

1. Arduino IDE Configuration:

- Install the Arduino IDE and open it.
- Go to `Tools > Board` and select "Arduino Nano 33 BLE".
- Ensure the correct port is selected under `Tools > Port.`

2. Library Installation:
   - Go to `Tools > Manage Libraries...` in the Arduino IDE.
   - Search for and install the PDM library for microphone input handling.

Test Procedures

1. **Digital Microphone Test:** The digital microphone on the Arduino Nano 33 BLE Sense is a key component for voice recognition. This test ensures that it can capture sound accurately.

   Objective: To verify the microphone's ability to capture and process audio signals.

   Procedure:

   a) Load the Test Sketch:
      - In the Arduino IDE, open the PDM library's example sketch designed for audio capture.
      - Review the sketch to understand how audio data is captured and processed.

   b) Upload the Sketch:
      - Connect the Arduino Nano 33 BLE Sense to your computer using the USB cable.
      - Upload the example sketch to the board.

   c) Conduct the Test:
      - Open the Serial Monitor in the Arduino IDE to view the output.
      - Generate a consistent sound toward the microphone (e.g.,YES or NO).
      - Observe the changes in the Serial Monitor output.

   Expected Outcome:

   - The Serial Monitor displays fluctuating numerical values or a waveform pattern that corresponds to the sound input.
   - Consistent sounds should produce recognizable patterns or consistent data outputs.

2. **Built-in LED Test:** The built-in LED on the Arduino Nano 33 BLE Sense provides visual feedback, which is crucial for signaling the detection of specific keywords.

   Objective: To confirm that the built-in LED can display various colors and respond to voice commands.

   Procedure:

   a) Prepare the LED Test Sketch:

      - Write or modify a simple sketch that sequentially lights up the LED in different colors (red, green, blue) with a pause between each color.
      - Include commands to control the LED's brightness if necessary.

   b) Upload the Sketch:

      - With the Arduino connected, upload your LED test sketch to the Nano 33 BLE Sense.

   c) Observe the LED Behavior:

      - Watch the LED as the sketch runs. Note the color changes and any variations in brightness.

   Expected Outcome:

   - The LED cycles through the specified colors clearly and smoothly.
   - Adjustments in brightness are noticeable if included in the test sketch.

3. **Reset Functionality Test:**

   Objective: Ensure the reset button correctly reboots the board without issues.

   Procedure:

   - While the Arduino Nano 33 BLE Sense is connected and running any sketch, press the reset button on the board.
   - Observe the behavior of the board and any connected peripherals or indicators (e.g., LEDs).

   Expected Outcome: The board resets and restarts the sketch from the beginning, as indicated by the operational sequence of LEDs or serial output, confirming the reset functionality.

Troubleshooting

- If any component does not perform as expected, check connections, ensure correct board selection in the Arduino IDE, and verify that all necessary libraries are installed and up to date.
- For the digital microphone test, ensure no other program is using the Serial Monitor and adjust the volume of the sound source if necessary.

Conclusion

This document provides a structured approach to testing the critical hardware components of the Arduino Nano 33 BLE Sense in the context of a keyword spotting project. Successful completion of these tests confirms the hardware's readiness for development and deployment of the application.

3.5. Data Quality in Hardware Description

Within the hardware description of our project utilizing the Arduino Nano 33 BLE Sense, we delve into several facets related to data quality:

1. **Precision of Sensors:**

   The integrated sensors on the Arduino Nano 33 BLE Sense board deliver precise measurements of environmental parameters, encompassing motion, orientation, temperature, humidity, and pressure.

2. **Sampling Rate:**

   The sensors' sampling rate determines the frequency of data collection. The Arduino Nano 33 BLE Sense board supports high sampling rates, facilitating real-time data acquisition at <some frequency> Hz.

3. **Noise Mitigation:**

   Despite potential noise sources like sensor imperfections and external interference, the data's noise level is minimized through adept calibration and filtering techniques.

4. **Efficient Data Transmission:**

   Sensor data is efficiently transmitted from the Arduino Nano 33 BLE Sense board to other devices through wireless communication protocols like Bluetooth Low Energy (BLE), ensuring minimal latency and reliable delivery.

5. **Data Integrity Assurance:**

   Measures are taken to guarantee the integrity of sensor data during transmission and processing. Employing error detection and correction mechanisms helps identify and rectify instances of data loss or corruption.[**TensorFlow:2023**]

6. **Optimized Power Consumption:**

   The Arduino Nano 33 BLE Sense board exhibits low power consumption, rendering it suitable for battery-powered applications. Power requirements are meticulously managed to extend battery life while sustaining continuous data acquisition.

3.5.1. Specifications of Arduino Nano 33 BLE Sense

NINA B306 Module

1. Processor:
   - 64 MHz Arm® Cortex-M4F (with FPU)
   - 1 MB Flash + 256 KB RAM

2. Bluetooth® 5 multiprotocol radio:
   - 2 Mbps
   - +8 dBm TX power
   - -95 dBm sensitivity
   - 4.8 mA in TX (0 dBm)
   - 4.6 mA in RX (1 Mbps)
   - Integrated balun with 50 Ohm single-ended output
   - IEEE 802.15.4 radio support

3. Peripherals:
   - 12 Mbps USB
   - NFC-A tag
   - Arm CryptoCell CC310 security subsystem
   - QSPI/SPI/TWI/I2S/PDM/QDEC
   - 32 MHz SPI
   - Quad SPI interface 32 MHz
   - 12-bit 200 ksps ADC
   - 128 bit AES/ECB/CCM/AAR co-processor

3.6. Constraints

Arduino Nano 33 BLE Sense

- Operating Voltage: 3.3V
- Power Consumption:
  - Maximum 15mA in low power mode
  - Maximum 60mA in active mode
- Operating Temperature Range: -40°C to 85°C
- Memory Constraints: 1 MB Flash + 256 KB RAM
- Communication Interfaces: USB, Bluetooth 5, NFC-A, SPI, I2C, QSPI, etc.

Actuators

- RGB LEDs:
  - Power Requirements: Voltage, Current
  - Operating Temperature Range
- Buzzer/Speaker:
  - Voltage, Current, Sound Output Levels

Power Supply

- Input Voltage Range: Specify the acceptable input voltage range.
- Power Consumption: Estimate the overall power consumption of the system.

Physical Constraints

- Size and Dimensions: Ensure compatibility with the project enclosure or housing.
- Mounting Requirements: Specify any specific mounting requirements for the components.

Environmental Constraints

- Environmental Protection: Ensure components are suitable for the intended environmental conditions (e.g., moisture resistance, dust resistance).
- Operating Conditions: Specify any limitations or special considerations for operating in certain environments.

3.7. Dimensions of the Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense is a compact board with dimensions of 45mm x 18mm. This small form factor makes it ideal for wearable devices and compact projects. However, despite its small size, it comes packed with a variety of sensors and features, making it a versatile choice for many projects. Please note that these dimensions are for the board without headers. If you're using a version with headers or if you're adding additional components to the board, this could affect the overall dimensions of your hardware setup. Always refer to your specific board and components for the most accurate dimensions. [Ard12]

4. Software Description

To implement a TinyML project for keyword spotting using an Arduino Nano 33 BLE Sense, several software components and tools are necessary. Here's an overview:

4.1. Arduino IDE Description

It is an open source official Arduino software which used for editing, uploading and compiling codes in to the Arduino module. It is a cross-platform software which is available for Operating Systems like Windows, Linux, macOS. It runs on Java platform and supports a range of Arduino modules. It supports C and C++ languages. The microcontrollers present on the Arduino boards are programmed which accepts the information in the form of code. The program written in the IDE is called a sketch which will generate a Hex file which is then transferred and uploaded in the controller. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module.[FAD18] The Menu bar has options such as File in which there are many options including Opening a new file or existing, Examples-in which we can find sketches for different applications like Blink, Fade etc. There is an error console at the bottom of the screen for displaying errors.

The 6 buttons are present on top of the screen are as follows:

Figure 4.1.: **Menu Button.**

- The check mark is used to verify your code. Click this once you have written your code.
- The arrow uploads your code to the Arduino to run.
- The dotted paper will create a new file.
- The upward arrow is used to open an existing Arduino project.
- The downward arrow is used to save the current file.

- The far right button is a serial monitor, which is useful for sending data from the Arduino to the PC for debugging purposes.

4.1.1. Installation

To install the Arduino IDE, we need to download the latest version from the Arduino webpage `https://www.arduino.cc/en/software`. We can select the version based on the operating system we are using. Here we are installing Arduino 1.8..15 for a Windows 10 operating system. The set up file name is arduino-1.8.15-windows.exe and the size of it is 1,17,470 KB. we can specify the path according to our needs. Here the path is set as C:/Program Files (x86)/Arduino.

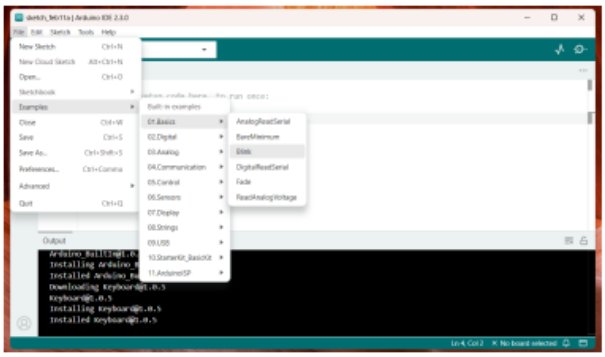


Figure 4.2.: **Menu Bar Option**

After the download is done, open the setup file and proceed to install. Select all the components in the dialog box and click Next.

Select the destination folder and click Install

Once the installation is done, open the Arduino IDE and a default sketch appears on the screen as shows.

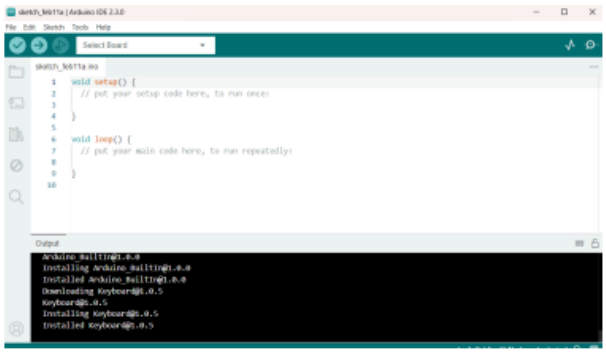


Figure 4.3.: **ArduinoIDESketch**

It can be seen from the above figure that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns

void and we do the intiliaztion such as the output LED color, specifying the core etc. The second part is the function `void loop()` where we define functions which are to be performed through out the loop. These codes are placed between paranthesis {} and each function has a return type, here it has void return type.

4.1.2. Arduino IDE on PC

Installation

Arduino Nano 33 BLE Sense uses the Arduino software integrated development environment (IDE) for programming, which is the most widely used and common (IDE) for all arduino boards that can be run online and offline. This is a open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. There are various version of software which is supported for each operating system (OS) e.g: mac, linux, and windows. Arduino community also provide us to start coding online and save our sketches in the cloud, this online arduino editor is most up-to-date version of the IDE includes all libraries and also supports new Arduino boards. For getting access to these software packages go to the following link `https://www.arduino.cc/en/software` and get more up to date inforamtion, because every single day there are some updates occurs which is available on the link mention above. These software can be used with any Arduino board, the most recent offline arduino IDE 1.8.15 can be seen in Figure,**??**. it is also supportive for all operating systems.

Figure 4.4.: **Arduino Creat Agent Installation**

4.1.3. Configuration

Configuration for the Arduino Nano 33 BLE Sense

To program the Arduino Nano 33 BLE Sense in offline state, we need to install one of the latest arduino IDE on our desktop. After installation, for getting access to the Arduino nano 33 ble sense board, we need to make configuration in our IDE. By opening the IDE, go to tool which can

be seen on the uper left corner in IDE, in the tool there is an option for managed board. At this point we need to write our board name in the search which is Arduino Nano 33 BLE Sense as shown in figure,4.5.Select the Arduino Mbed OS Boards and install it. The Mbed OS nano board supports also other nano family boards including Arduino nano 33 ble sense, after installing simply connect the Arduino Nano 33 BLE Sense to the computer via USB cable.

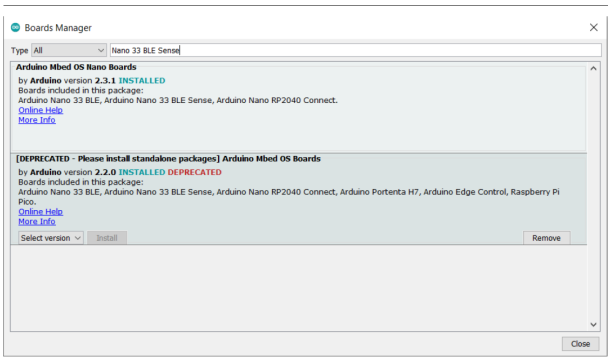


Figure 4.5.: **Arduino Mbed OS Nano Boards Installation**

4.1.4. Setup

There are set of examples which are build in Arduino (IDE) for the testing purpose, for checking all the configuration and setting up the board we can open one of the basic LED blink example first as shown in the figure.

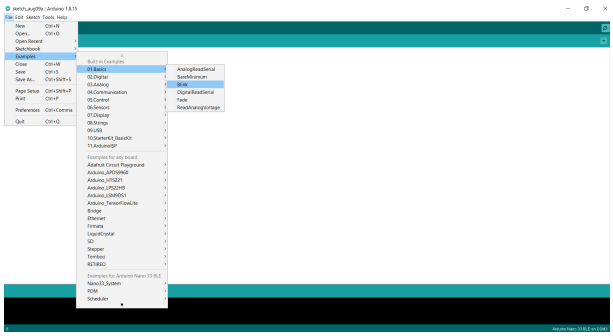


Figure 4.6.: **Setup Test**

This LED-blink example support all the arduino boards, for the checking purposes just need to run this basic example on any arduino embed board and it will blink the LED on our Arduino board after pre-set miliseconds. In the same example folder, there are also number of build in usefull example written in Arduino IDE for embedded boards. These examples are very usefull for getting the basic knowledge about the board and programming.

4.1.5. constraints

There are some pre-requisite steps need to follow either we need to run the build in example or run by our own written program. By operating the Arduino board with Laptop with the help of USB connection, need to open the Arduino IDE on desktop, it appears a blank arduino environment page just a Void setup and void loop written on it. At this step we need to go to the tool-Arduino board and select the connected board which is Arduino Nano 33 Ble Sense as shown in the figure.

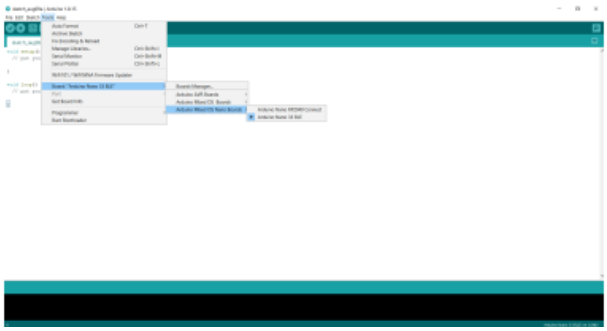


Figure 4.7.: **Select the Connected board here Arduino Nano 33**

Select the Appropriate Port

By selecting the Arduino nano 33 BLE sense board, next we need to check the connected port. For doing this, we need to set our arduino borad in Boot setup by clicking the white reset button on arduino as show in figure.

Figure 4.8.: **Arduino Nano 33 BLE Sense Reset Button**

After successfully applying the above mention step, next we need to select the connected port before upload the program. For this, go to tool select arduino port and make sure to check it available port for uploading the program as shown in figure**??**

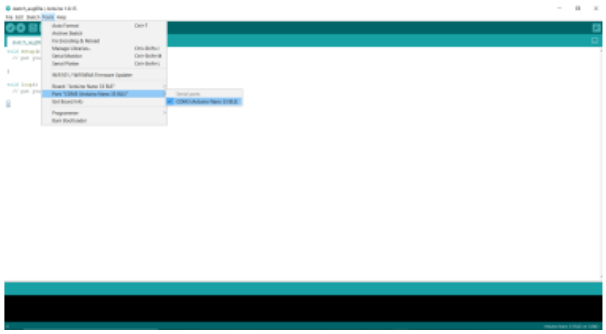


Figure 4.9.: **Select Available Port for Up loading Arduino Sketch**

4.1.6. Data quality

The Arduino Integrated Development Environment (IDE) is a key component in the Magic Wand project with Arduino Nano 33 BLE Sense. Here's a detailed explanation of its quality aspects.[FAD18]

1. **Ease of Use**: The Arduino IDE is user-friendly and designed to be easy to use for beginners, while also providing advanced features for experienced users.

2. **Cross-Platform Compatibility**: The Arduino IDE is compatible with Windows, macOS, and Linux, making it accessible to a wide range of users.

3. **Open-Source**: The Arduino IDE is open-source, which means its source code is freely available. This allows for community contributions, leading to continuous improvements and updates.

4. **Support for Multiple Boards**: The Arduino IDE supports not only the Arduino Nano 33 BLE Sense but also a wide range of other Arduino boards.

5. **Built-In Code Editor**: The Arduino IDE comes with a built-in code editor that provides features like syntax highlighting and automatic indentation, making it easier to write and debug code.

6. **Library Manager**: The Arduino IDE includes a Library Manager that makes it easy to install and manage libraries. This is particularly useful for the Magic Wand project, which requires libraries like TensorFlow Lite for Microcontrollers.

7. **Serial Monitor**: The Arduino IDE includes a Serial Monitor that allows you to monitor data sent from the Arduino Nano 33 BLE Sense in real-time. This is crucial for the Magic Wand project, as it allows you to monitor the gesture recognition process.

8. **Community Support**: The Arduino IDE has a large and active community. This means you can find a wealth of tutorials, guides, and troubleshooting advice online.

4.1.7. Data quantity

Magic Wand project with Arduino Nano 33 BLE Sense, "Data Quantity" in the software description of the Arduino IDE refers to the amount of data that the software can handle or process. Here's a detailed explanation:

1. **Code Size**: The Arduino IDE needs to handle the code for the keyword spotting project. This includes the code for collecting and processing sensor data, running the machine learning model, and any additional functionality.

2. **Library Size**: The Arduino IDE also needs to handle various libraries that are used in the project. This includes the TensorFlow Lite for Microcontrollers library, which is used to run the machine learning model on the Arduino Nano 33 BLE Sense.

3. **Sensor Data**: The Arduino IDE needs to handle the sensor data that is collected by the Arduino Nano 33 BLE Sense. In one experiment, a window size of 2 seconds was used, which means 200 rows of accelerometer data or 600 values of x, y, and z acceleration axis were fed into the model.[FAD18]

4. **Model Data**: The Arduino IDE needs to handle the data of the machine learning model. This includes the model parameters and the model output.

5. **Data Transmission**: The Arduino IDE also handles data transmission between the Arduino Nano 33 BLE Sense and the computer. This includes uploading the program to the board and transmitting sensor data and model output to the computer for monitoring.

4.1.8. Data types

- **Software Components:**
  - **TensorFlow Lite For Microcontrollers:** This is an optimized version of TensorFlow, targeted to run TensorFlow models on tiny, low-powered hardware such as microcontrollers. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.
  - **Arduino IDE:** This is the software used to write and upload computer code to the physical board.

- **Data Types:** You'll likely be working with arrays to store the sensor data, and you'll use TensorFlow Lite's data types for the model's input and output. The exact data types will depend on your specific implementation and the requirements of the TensorFlow Lite model.

1. **Numeric Data Types:**
   - Identify the numeric data types used, such as integers, floating-point numbers, or doubles.
   - Describe the range and precision of each numeric data type.
2. **Text Data Types:**
   - Specify if the software handles text data, such as strings or characters.
   - Discuss any character encoding or formatting requirements.
3. **Sensor Data Types:**
   - If interacting with sensors, specify the types of sensor data used.
   - Describe the format and units of measurement for each sensor data type.
4. **Custom Data Types:**
   - Discuss any custom data types defined in the software, such as structs or classes.
   - Explain the purpose and structure of each custom data type.
5. **Data Conversion and Casting:**
   - If data conversion or casting operations are performed, describe how different data types are converted or casted.
6. **Data Representation:**
   - Explain how data is represented, including binary, hexadecimal, or other representations.

4.1.9. Data structure

- **Data Structures:**
  - **Sensor Data:** The Arduino Nano 33 BLE Sense collects sensor data, which is multidimensional and complex. This data is typically stored in arrays or similar data structures for processing.

- **Model Input and Output:** The sensor data is passed to the TensorFlow Lite model as input, and the model outputs a simple classification. The exact data structures for these will depend on the requirements of your TensorFlow Lite model.[Har23]

1. **Arrays:** Used to store a collection of data items of the same type. Example: storing sensor readings over time, storing LED brightness levels.

2. **Structures (`struct`):** Allows grouping multiple variables of different types under a single name. Example: defining a struct to hold sensor data (e.g., temperature, humidity).

3. **Classes:** Used for creating user-defined data types with properties and methods.

4. **Linked Lists:** Data structures consisting of a sequence of elements where each element points to the next element. Example: implementing a linked list to manage a queue of actions to perform.

5. **Stacks:** Last in, first out (LIFO) data structures where elements are inserted and removed from the same end. Example: using a stack to manage function calls or nested operations.

6. **Queues:** First in, first out (FIFO) data structures where elements are inserted at the end and removed from the front. Example: implementing a queue for handling incoming commands or tasks.

7. **Trees:** Hierarchical data structures consisting of nodes connected by edges. Example: representing hierarchical data such as menu structures or organizational charts.

4.1.10. Conclusions

Output Window (Serial Monitor)

Serial Monitor is the another window on the Arduino IDE, which shows the Input/Output of our program and results appear on it as per the required output. For getting access to Serial monitor, we need to go extreme right in the Arduino IDE, the small circle pop up when we reach it is the serial monitor as show in the figure.

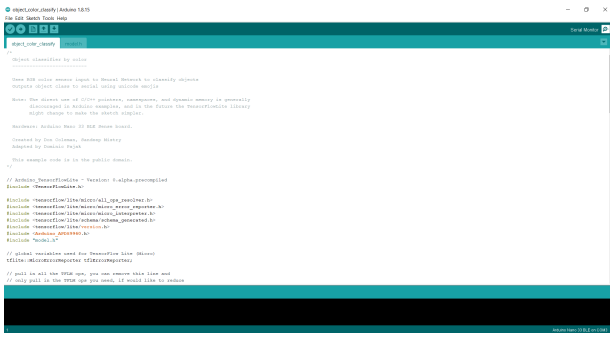


Figure 4.10.: **Serial Monitor**

The Final results, all the variables, input, sensor values are shown in the serial monitor the (Output Window) as shown in the figure by clicking the serial monitor button.

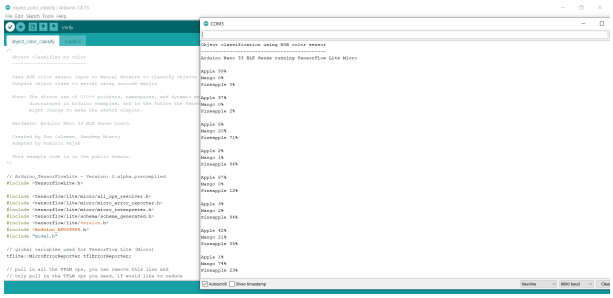


Figure 4.11.: **Output Window**

4.2. TensorFlow

TensorFlow is an end-to-end, open-source platform popularly used for the quick implementation of machine learning algorithms. A rich ecosystem of tools, libraries, and community resources has made it extremely popular among machine learning researchers and practitioners to develop and deploy various machine learning algorithms with greater efficiency and flexibility. TensorFlow has become popular in recent times for the quick development of complex deep neural network architectures for both experimentation and developing production-ready software.[Kha21]

TensorFlow was originally developed by Google within their Machine Intelligence Research Organization to conduct various machine learning and neural networks related research. The initial version was released in 2015 under the Apache License 2.0. TensorFlow 2.0, the newest stable version, was released in 2019. TensorFlow is highly flexible. It supports a wide variety of programming languages, including Python, C++, and Java. Moreover, it can run on CPU, GPU, and TPU for a faster processing of large machine learning applications. TensorFlow is available in Linux,

macOS X, and Windows platforms. It also supports TensorFlow Lite, a highly optimized lighter version of the original TensorFlow that is available on mobile computing platforms such as Android, iOS-based smartphones, and Linux-based single board computers like Raspberry Pi. TensorFlow Lite models can be further optimized using a few standard APIs to run on microcontroller units. Hence, TensorFlow is heavily used in TinyML applications. Visit the official TensorFlow website for more details. To summarize, some of the key features of TensorFlow are as follows:

- It is open-source
- Efficiently works with multi-dimensional data
- Provides a higher level of abstraction, which reduces the code length for the developer
- Supports various platforms and architectures
- Highly scalable and provides greater flexibility for quick prototyping

4.2.1. Installation

TensorFlow, along with all its dependencies, is already installed in Colab. So, you just need to import the libraries to write codes without any package installation. Once TensorFlow is imported, you can check the version by typing the command **tf.__version__**. At the time of writing this book, the TensorFlow version in Colab is 2.12.0, which may change with time. TensorFlow 2, or TF 2, is the newest version which is significantly different compared to the previous version TF 1.x. In this book, we will use TF2 for all our programming. However, TF 2 provides a backward compatibility module to use TF 1.x. The eager execution mode of TF 2 makes it easier to create a machine learning architecture with lesser lines of code.

4.2.2. Data Quality

- **Data Collection:** The Arduino Nano 33 BLE Sense collects multidimensional sensor data. The quality of this data is important. Ensure that the sensor is working correctly and that the data is being collected under appropriate conditions.
- **Data Preprocessing:** The raw sensor data may need to be preprocessed before it can be used as input to the TensorFlow Lite model. This could involve normalizing the data, dealing with missing values, or other preprocessing steps. The quality of the preprocessing can significantly affect the model's performance.

- **Model Training:** The TensorFlow Lite model is trained on a dataset. The quality of this training data is crucial. It should be diverse and representative of the different gestures that the "magic wand" should recognize.

- **Model Testing:** The model should be tested on a separate dataset to ensure that it generalizes well to new data. The quality of this testing data is also important.

4.2.3. Data Quantity

In the Magic Wand project with Arduino Nano 33 BLE Sense, the quantity of data is a crucial aspect as it directly impacts the performance of the TensorFlow Lite mode

- **Sensor Data:** The Arduino Nano 33 BLE Sense collects multi-dimensional sensor data. The quantity of this data is important. More data can lead to better model performance, but it also requires more storage and processing power.

- **Model Training:** The TensorFlow Lite model is trained on a dataset. The size of this training data is crucial. Larger datasets can lead to better model performance, but they also require more computational resources.

- **Model Testing:** The model should be tested on a separate dataset to ensure that it generalizes well to new data. The size of this testing data is also important.

4.2.4. Data Types

- **Sensor Data:** The Arduino Nano 33 BLE Sense collects multidimensional sensor data. This data is typically stored in arrays or similar data structures for processing.

- **Model Input and Output:** The sensor data is passed to the TensorFlow Lite model as input, and the model outputs a simple classification. The exact data types for these will depend on the requirements of your TensorFlow Lite model.

- **Gesture Recognition:** The project involves recognizing gestures, which are classified into different categories. You might use data structures like enumerations or similar to represent these different categories.

4.2.5. Data Structure

In the software description section of project involving TensorFlow, need to specify the data structures used in your software.

1. **Tensor:** The fundamental data structure in TensorFlow is the tensor, which is a multi-dimensional array or matrix. Tensors can have different ranks, representing scalars, vectors, matrices, or higher-dimensional arrays.

2. **Graph:** TensorFlow uses a computational graph to represent the flow of data and operations in a machine learning model. The graph consists of nodes that represent mathematical operations and edges that represent the flow of tensors between these operations.

3. **Variables:** Variables are tensors whose values can be modified during the execution of a TensorFlow program. They are commonly used to store the parameters of a machine learning model, such as weights and biases.

4. **Placeholder:** Placeholders are used to feed input data into a TensorFlow graph during execution. They serve as empty nodes that are filled with actual data at runtime.

5. **Dataset:** TensorFlow provides the `tf.data.Dataset` API for managing input data pipelines. Datasets represent sequences of elements, such as training examples or batches of data, and support operations like batching, shuffling, and prefetching.

6. **Layers:** TensorFlow's high-level API, `tf.keras`, includes a variety of predefined layers for building neural networks. These layers encapsulate common operations like convolution, pooling, and dense connections, making it easy to construct complex models.

7. **Optimizer:** Optimizers are used to minimize the loss function during training by updating the parameters of the model. TensorFlow provides a range of optimizers such as stochastic gradient descent (SGD), Adam, and RMSprop.

8. **Callbacks:** Callbacks are utility functions in TensorFlow that can be used to perform actions at various stages of training, such as saving model checkpoints, logging metrics, or early stopping based on validation performance.

4.2.6. Constraints

Once we are done with loading and pre-processing of the data, we can define neural network architecture.

A Keras model has the following four stags:

1. Defining the model: We define a Sequential model and add the necessary layers.

2. Compiling the model: We configure the model for training by defining the loss function to be minimized, the optimizer that minimizes the loss function, and a performance metric to internally evaluate the performance.

3. Fitting the model: Here, we fit the model on the actual training data for a given number of epochs to update the training parameters. We will get the model at the end of training.

4. Evaluation: Once you have the model, you can evaluate on unseen test data

4.2.7. Conclusions

One of the biggest benefits of using TensorFlow is the level of abstraction. It takes care of the major details of most of the underlying algorithms in a machine learning or a deep learning application, for example, back-propagation. Hence, writing programs in TensorFlow is super easy as you mostly need to focus on your application logic. TensorFlow applications can run on almost any target environment, such as local desktops, remote servers running Windows, Linux, macOS X, smartphone devices running Android and iOS, or Linux-based single board computers. TensorFlow contains additional libraries to convert a machine learning model into C++ equivalent libraries to run on selected microcontrollers. Hence, you can use TensorFlow to create your TinyML applications. Throughout this book, we will primarily use Keras to implement the machine learning models. Keras is a high-level set of Python Application Programming Interface (API) in TensorFlow, which is popularly used in the rapid prototyping of various neural network models. Fortunately, both TensorFlow and Keras are readily available in Colab. So, you can easily start writing your program in Colab without installing any extra libraries. In this chapter, we will primarily focus on creating end-to-end neural network models using TensorFlow. The later chapters will focus on optimizing the neural networks to create deployable TinyML applications.

4.3. Python

The Arduino IDE is written only in C++ language and is not supported the other programming languages. Some of the Module I used specially for executing the Gesture detection part of this project is only support python language e.g., MediaPipe and OpenCV. MediaPipe and OpenCV module are the most important part for gesture detection, for detecting the landmarks on hand the supported function is written in python. Without the MediaPipe Module, the hand landmarks technique is not possible to implement. At the moment, there is no such module or library who can support directly MediaPipe Module in Arduino IDE and C++, because MediaPipe Module is written in python.

4.3.1. Installation

```
1000
1001  import tensorflow as tf
1002  import numpy as np
1003
1004  # Load the TensorFlow Lite model.
      interpreter = tf.lite.Interpreter(model_path="model.tflite"
        )
1006  interpreter.allocate_tensors()
1007
1008  # Get the input and output tensors.
      input_details = interpreter.get_input_details()
1010  output_details = interpreter.get_output_details()
1011
1012  # Create some input data.
      input_data = np.array([[1.0, 2.0, 3.0]])
1014
1015  # Set the input tensor.
1016  interpreter.set_tensor(input_details[0]["index"],
        input_data)
1017
1018  # Perform inference.
      interpreter.invoke()
1020
1021  # Get the output tensor.
1022  output_data = interpreter.get_tensor(output_details[0]["
        index"])
1023
1024  # Print the output.
      print(output_data)
1026
```

4.3.2. Data - Quality , Type, Structure

Data Quality:

- **Data Validation:** Describe the methods used for data validation in the Python code. This may include input validation, error handling, and data cleaning techniques to ensure the quality and integrity of the data processed by the program.

- **Testing and Debugging:** Discuss the testing and debugging procedures implemented to verify the correctness and reliability of the Python code. This includes unit tests, integration tests, and debugging tools used during development.

- **Error Handling:** Explain the error handling mechanisms incorporated into the Python code to handle exceptions, edge cases, and unexpected behaviors gracefully. Describe how errors are logged and reported to facilitate troubleshooting.

Data Type:

- **Dynamic Typing:** Highlight the dynamic typing feature of Python, which allows variables to change data types dynamically during runtime. Discuss the flexibility and convenience provided by dynamic typing in programming tasks.

- **Built-in Data Types:** Describe the built-in data types supported by Python, including integers, floats, strings, lists, tuples, dictionaries, and sets. Explain their characteristics, usage, and typical scenarios where each data type is preferred.

Data Structure:

- **List and Tuple:** Discuss the list and tuple data structures in Python, their properties, and differences. Explain how lists and tuples are used to store and manipulate ordered collections of elements.

- **Dictionary:** Describe the dictionary data structure in Python, which stores key-value pairs. Explain how dictionaries are used for efficient data retrieval and mapping between keys and values.

- **Set:** Explain the set data structure in Python, which stores unique elements without any specific order. Discuss the operations supported by sets, such as union, intersection, and difference.

4.3.3. Constraints

Python, although versatile, has constraints in the context of the magic wand project with Arduino Nano 33 BLE. Real-time operations pose a challenge due to Python's lack of native support, potentially affecting precise timing for gesture recognition. Additionally, Python's interpreted nature may result in slower execution, impacting sensor data processing and model inference speed. Memory usage can be a concern on resource-constrained microcontrollers, like Arduino Nano. While Python is not inherently optimized for the lightweight requirements of such devices, developers can mitigate these constraints by employing efficient coding practices and considering alternatives for critical real-time tasks.

4.3.4. Conclusion

The use of Python in the software aspect of the magic wand project enhances flexibility, allowing for efficient data processing, machine learning model training, and seamless communication with the Arduino Nano 33 BLE.

4.4. PyCharm

PyCharm, a robust Python integrated development environment (IDE), plays a pivotal role in the software development process for the magic wand project with Arduino Nano. Leveraging PyCharm's advanced code editing and navigation features, the development team benefits from an efficient and organized coding experience. The IDE's support for project management, version control integration with Git, and seamless debugging and profiling tools contribute to a streamlined development workflow. PyCharm's capabilities extend to virtual environment support, ensuring effective dependency management for the machine learning aspects of the project. Additionally, PyCharm facilitates integration with continuous integration (CI) systems, enabling automated testing and build processes. Overall, PyCharm proves indispensable in enhancing code quality, collaboration, and the overall efficiency of the machine learning software development on the Arduino Nano platform.

4.4.1. Setup

To install PyCharm, begin by visiting the official PyCharm website and downloading the appropriate version for your needs—Community (free) or Professional (paid). Run the installer, typically an executable file on Windows, and follow the on-screen instructions. Choose the installation location, edition, and any additional settings. If you opt

for the Professional edition, activate a license during the installation or choose to evaluate it for a trial period. Once installed, run PyCharm, configure a Python interpreter, and start coding. It's advisable to consult the official PyCharm documentation for detailed and platform-specific instructions.

4.4.2. Constraints

PyCharm, while a versatile and powerful integrated development environment (IDE) for Python, presents certain constraints that users should consider. One notable constraint is its resource intensiveness, potentially causing performance issues on machines with limited RAM or processing power, especially in larger projects. Additionally, the learning curve can be steep for new users due to the extensive feature set. Cost can be a constraint for users requiring advanced features in the professional version. Compatibility issues may arise with specific Python libraries or frameworks, and users should stay updated to avoid such constraints. While PyCharm is specialized for Python, its support for other languages may be limited. Plugin dependencies, heavy initial indexing for large projects, and constraints related to remote development or extremely complex projects further warrant consideration. Awareness of these constraints empowers users to make informed decisions based on their development needs and project characteristics.

4.4.3. Conclusion

PyCharm serves as an integral component of the development toolkit for the magic wand project with Arduino Nano. Its comprehensive set of features enhances the coding experience, fosters collaboration, and contributes to the overall efficiency of the software development process.

5. Data Mining: Convolutional Neural Network (CNN)

5.1. Introduction

Over recent years, there has been notable advancement in the realm of artificial intelligence, particularly in the domain of deep learning. Presently, deep learning stands as a pivotal element in numerous cutting-edge technological applications, ranging from autonomous vehicles to the creation of art and music. The scientific community is focused on empowering computers not only to comprehend speech but also to articulate in a manner akin to natural languages. Deep learning, as a subset of machine learning, distinguishes itself by its emphasis on learning data representation instead of relying on task-specific algorithms. This approach allows computers to construct intricate concepts by synthesizing information from simpler and more elementary components [SKP18; Li+21].

Building on the insights presented by Warden [WS19] and Giménez [LG+22], Convolutional Neural Network (CNN) is chosen as the training model, making it suitable for speech recognition applications.

A convolutional neural network (CNN) stands out as a distinct type of multi-layer neural network tailored for direct recognition of visual patterns in images with minimal processing. In the realm of artificial neural networks, a neuron functions as a transformative entity, taking an input and producing an output. The quantity of neurons employed is contingent upon the specific task, ranging from as few as two to several thousand. Various methods exist for interconnecting artificial neurons to formulate a CNN, where each neuron receives inputs from others, and the impact of each input is regulated by its associated weight, which can be either positive or negative. The collective learning process of the neural network facilitates the execution of valuable computations essential for object recognition through language comprehension. The interconnection of these neurons culminates in the creation of a feed-forward network, where the output from each layer of neurons is propelled forward to subsequent layers, ultimately yielding a conclusive output [SKP18; Gu+18].

This chapter begins with an overview of the CNN algorithm, followed by a discussion on its applications. The rationale behind selecting this algorithm is then highlighted, and the ensuing section delves into the algorithm's requirements. Subsequently, the inputs and outputs of the

algorithm are examined, presented both in a general context and specifically within the framework of keyword spotting. The chapter concludes with the inclusion of a Python example code.

5.2. Description

Understanding the inner workings of the model is not a prerequisite for its utilization, but delving into its mechanisms can prove beneficial for troubleshooting issues and holds intrinsic interest. This section offers insights into the model's predictive processes [WS19].

In the realm of neural network architecture, there exists a type specifically adept at handling multidimensional tensors, where information is embedded in relationships among adjacent values—a convolutional neural network (CNN). While commonly applied to images, which are 2D grids of pixels, CNNs exhibit remarkable efficacy in processing spectrogram data, showcasing their versatility beyond conventional visual inputs [WS19].

In this section, the fundamentals of CNN are introduced. Additionally, explanations for essential components such as the activation function, loss function, and optimizer are provided.

5.2.1. Description of Basic CNN Components

CNNs, a type of feedforward neural network, excel in automatically extracting features from data through convolutional structures. In contrast to traditional feature extraction methods, CNNs eliminate the need for manual feature extraction. Inspired by visual perception, the CNN architecture aligns artificial neurons with biological neurons, employing kernels to represent receptors responding to various features. Activation functions simulate neural electric signal transmission, akin to the biological threshold for signal transmission between neurons. Loss functions and optimizers are designed to guide the CNN system in learning desired patterns [Li+21].

Compared to fully connected (FC) networks (see Figure 5.1), CNNs offer several advantages:

- **Local connections:** Neurons are no longer connected to all neurons of the previous layer but only to a small number, reducing parameters and accelerating convergence.
- **Weight sharing:** Groups of connections share the same weights, further reducing parameters.
- **Downsampling dimension reduction:** Pooling layers leverage image local correlation principles to downsample, reducing data

volume while retaining essential information and eliminating trivial features.

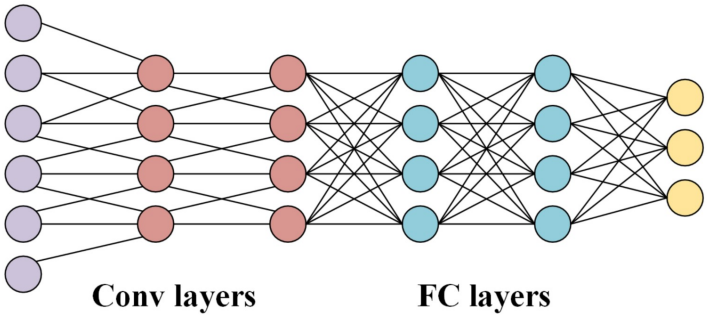


Figure 5.1.: CNN and FC layers [Li+21].

Convolution is a crucial step for feature extraction. Convolution generates feature maps. To mitigate information loss at the border, padding enlarges the input with zero values. Stride is employed to control convolution density. Post-convolution, feature maps may lead to overfitting, necessitating pooling (max pooling or average pooling) for redundancy elimination. The overall CNN procedure is illustrated in Figure 5.2. The terms padding, stride, and pooling are explained as follows:

- **Padding:** Padding involves adding extra border pixels to the input image before convolution. This is done by appending additional rows and columns of zero values around the image. The purpose of padding is to mitigate information loss at the borders during convolutional operations.

- **Stride:** Stride is the step size or the number of pixels by which the convolutional filter moves across the input image during the convolution operation. A larger stride reduces the spatial dimensions of the feature maps, leading to a more compact representation.

- **Pooling:** Pooling is a down-sampling operation applied after convolution. It reduces the spatial dimensions of the feature maps, helping to control the number of parameters in the network. Pooling is typically performed using operations such as max pooling or average pooling.

  - **Max Pooling:** Max pooling is a specific pooling operation where, for each region of the feature map, the maximum value is taken. This helps retain the most significant information from that region, providing a form of spatial summarization.

– **Average Pooling:** Average pooling is another type of pooling operation where, for each region of the feature map, the average value is computed. This operation helps reduce the spatial dimensions while maintaining a smoother and less detailed representation.

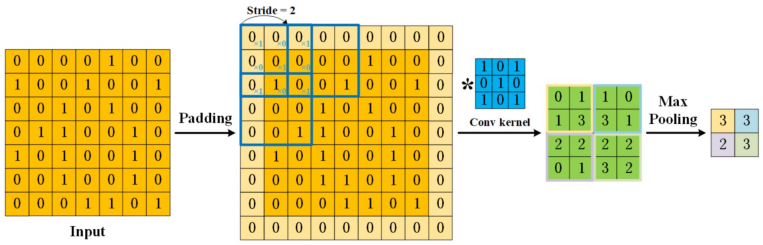


Figure 5.2.: Procedure of a two-dimensional CNN [Li+21].

In the realm of Convolutional Neural Networks (CNNs), a variety of architectural variations exists, yet their fundamental components remain highly similar. Taking the renowned LeNet-5 as a case in point, it comprises three key layers: convolutional, pooling, and fully-connected layers. The primary purpose of the convolutional layer is to capture feature representations from the inputs [Gu+18].

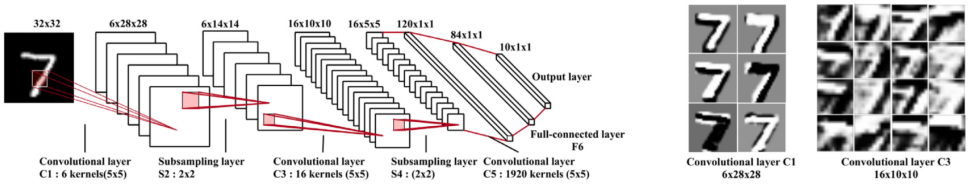
Illustrated in Figure 5.3a, the convolutional layer is crafted from multiple convolution kernels responsible for computing distinct feature maps. Each neuron within a feature map establishes connections with a neighborhood of neurons in the preceding layer, known as the neuron's receptive field. The generation of a feature map involves convolving the input with a learned kernel, followed by the application of an element-wise nonlinear activation function to the results. It's important to note that each feature map is generated by sharing the kernel across all spatial locations of the input, thereby leveraging several diverse kernels for the complete set of feature maps.

Mathematically, the feature value at location (i, j) in the kth feature map of the lth layer (denoted as $z^l_{i,j,k}$) is computed by the expression:

$$z^l_{i,j,k} = {\mathbf{w}^l_k}^T \mathbf{x}^l_{i,j} + b^l_k \tag{5.1}$$

Where $\mathbf{w}^l_k$ and b^l_k represent the weight vector and bias term of the kth filter of the lth layer, while $\mathbf{x}^l_{i,j}$ is the input patch centered at location (i, j) of the lth layer. The weight sharing mechanism, where the kernel generating the feature map is shared, offers advantages such as reducing model complexity and facilitating network training.

The activation function $(a(\cdot))$ introduces nonlinearity to the CNN, which is crucial for detecting nonlinear features in multi-layer networks.



(a) LetNet-5 network (b) Learned features

Figure 5.3.: The architecture of the LeNet-5 network [Gu+18]. (a) The architecture of the LeNet-5 network, renowned for its effectiveness in digit classification tasks. (b) Displaying the features within the LeNet-5 network through visualizations, where each layer's feature maps are showcased in distinct blocks.

The activation value ($a_{l,i,j,k}$) of the convolutional feature $z_{l,i,j,k}$ is computed as:

$$a^l_{i,j,k} = a(z^l_{i,j,k}) \tag{5.2}$$

Common activation functions include sigmoid, tanh, and ReLU. More information about activation functions is given in Subsection 5.2.2. The pooling layer, positioned between convolutional layers, aims to achieve shift-invariance by downsampling the feature maps' resolution. Each feature map in the pooling layer connects to its corresponding feature map in the preceding convolutional layer. The pooling operation, denoted as pool$(\cdot)$, is applied to each feature map $a^l_{m,n,k}$ with:

$$y^l_{i,j,k} = \text{pool}(a^l_{m,n,k}), \forall (m,n) \in \mathcal{R}_{ij} \tag{5.3}$$

Here, $\mathcal{R}_{ij}$ represents a local neighborhood around location (i, j). Common pooling operations include average pooling and max pooling. The learned feature maps, as demonstrated in Figure 5.3b for the digit 7, progressively capture hierarchical features, with early layers detecting low-level features like edges and curves, and subsequent layers encoding more abstract features.

Following the convolutional and pooling layers, one or more fully-connected layers may be present to perform high-level reasoning. These layers connect all neurons from the previous layer to every neuron in the current layer, generating global semantic information. Notably, a fully-connected layer can be replaced by a 1×1 convolution layer.

The final layer in CNNs is the output layer. For classification tasks, the softmax operator (see Subsection 5.8.2) is commonly employed. Alternatively, SVM can be combined with CNN features to address diverse classification tasks. Consider a set of N desired input-output relationships represented as $\{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}); n \in [1, \ldots, N]\}$. The parameters of a CNN,

denoted as θ, including weight vectors and bias terms, are optimized by minimizing an appropriate loss function defined for the specific task:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \ell(\theta; \mathbf{y}^{(n)}, \mathbf{o}^{(n)}) \tag{5.4}$$

Here, $\mathcal{L}$ represents the loss function, which quantifies the discrepancy between the predicted output ($\mathbf{o}^{(n)}$) and the actual target label ($\mathbf{y}^{(n)}$) for nth input data point $\mathbf{x}^{(n)}$. The symbol θ denotes all the parameters of the CNN, including weight vectors and bias terms. The function $\ell(\theta; \mathbf{y}^{(n)}, \mathbf{o}^{(n)})$ is the individual loss for a specific data point, measuring the dissimilarity between the predicted output and the true label. The goal during training is to minimize the average loss over all data points, achieved through techniques such as stochastic gradient descent. Additional details about loss functions are available in Subsection 5.2.3.

The training of a CNN involves global optimization, with the aim of finding the best set of parameters by minimizing the loss function. Stochastic gradient descent emerges as a common solution for optimizing CNN networks, providing an effective means of iterative parameter adjustment. Optimizing is explained in the Subsection 5.2.4.

5.2.2. Activation Function

Convolutional Neural Networks (CNNs) leverage various activation functions to express intricate features [Li+21]. Functioning akin to the human brain's neuron model, the activation function serves as a unit determining the transmission of information to the next neuron. In a multilayer neural network, an activation function exists between two layers, structurally depicted in Figure 5.4.

In Figure 5.4, x_i denotes input features, n features simultaneously input to neuron j, w_{ij} represents connection weight between input feature x_i and neuron j, b_j is the internal state (bias) of neuron j, and y_j is the neuron's output. The activation function, denoted as $f(\mathring{\text{u}})$, includes options like the sigmoid, tanh, rectified linear unit (ReLU), and others.

When no activation function or a linear function is used, the network output is a linear combination of inputs, limiting learning ability. Nonlinear activation functions, like sigmoid and tanh, are introduced to enhance the network's capability to fit data.

Some of the well-known actiovatoion functions are shown in the Figure 5.5. The sigmoid function maps a real number to (0, 1), suitable for binary classification. The tanh function maps to (-1, 1), aiding normalization. ReLU, with its advantages in learning speed, is preferred in deep networks. Leaky ReLU and PReLU address limitations of ReLU, reducing neuron inactivation. ELU improves convergence by having a negative output average.

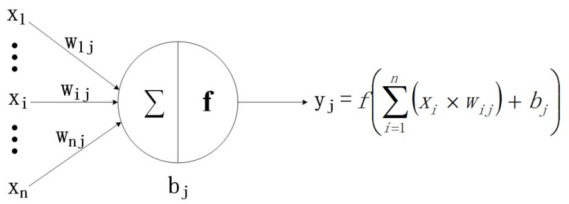


Figure 5.4.: Overall structure of an activation function [Li+21].

Swish, proposed by Google, and mish, a novel activation function, demonstrate improved performance compared to ReLU and Swish in deeper models across diverse datasets. mish, in particular, exhibits superior gradient flow, accuracy, and generalization properties. Experimental results consistently support the effectiveness of Mish across standard architectures.

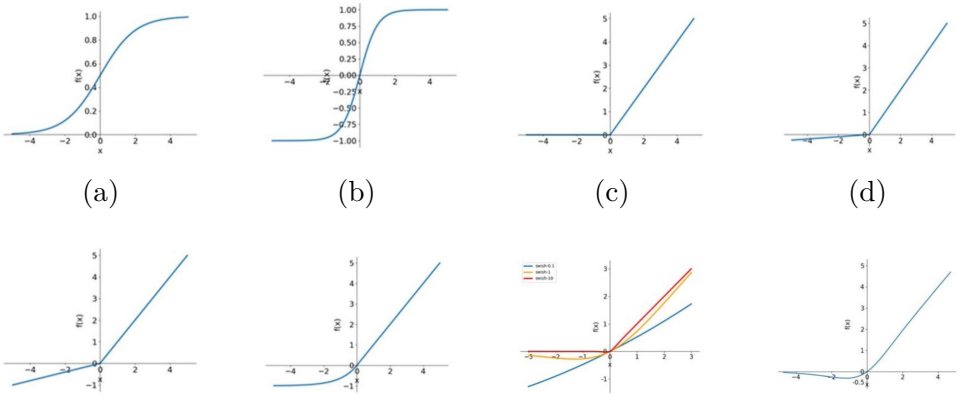


Figure 5.5.: Diagrams of activation functions [Li+21]. (a) Sigmoid function. (b) Tanh function. (c) ReLU function. (d) Leaky ReLU function. (e) PReLU function. (f) ELU function. (g) Swish function. (h) Mish function.

Guidelines for Activation Function Selection

1. For binary classification tasks, employ the sigmoid function in the last layer; for multiclassification, opt for the softmax function.

2. Exercise caution with sigmoid and tanh functions due to potential gradient vanishing issues. Preferably, in hidden layers, consider ReLU or leaky ReLU for improved performance.

3. When uncertain about the suitable activation function, consider experimenting with ReLU or leaky ReLU as reliable starting points.

4. In cases where numerous neurons become inactive during training, explore alternatives like leaky ReLU, PReLU, or similar activation functions to address the issue.

5. To expedite training, set the negative slope in leaky ReLU to 0.02, mitigating potential challenges associated with a slow convergence rate.

5.2.3. Loss Function

The loss function, crucial in calculating the disparity between predicted and actual values, serves as a learning criterion for optimization in CNNs. It plays a key role in regression and classification problems, aiming to minimize the loss. Commonly used loss functions include Mean Absolute Error (MAE), Mean Square Error (MSE), and Cross Entropy [Li+21].

Loss Function for Regression

In CNNs, MAE or MSE is commonly used for regression problems. MAE computes the mean absolute error, providing robustness to outliers. On the other hand, MSE calculates the mean of square errors, enabling controlled update rates. Choosing between them depends on the presence of outliers in the training set.

Loss Function for Classification

Various loss functions are employed in CNNs for classification tasks. The widely used cross entropy loss evaluates the difference between predicted probability distributions and actual distributions. While effective, it has limitations, focusing solely on classification correctness and neglecting factors like compactness within classes and margins between different classes.

- **Contrastive Loss:** Enlarges distance between different categories and minimizes distance within the same categories, often used in dimensionality reduction for face recognition.

- **Triplet Loss:** Introduced for better face embeddings, minimizes the distance between anchor and positive images while increasing the distance to negative ones. Useful in fine-grained classification at the individual level.

- **Center Loss:** An improvement upon cross entropy, focuses on the uniformity of distribution within the same class by minimizing intraclass differences.

These diverse loss functions offer flexibility and cater to specific challenges encountered in regression and classification tasks within CNNs.

Guidelines for Loss Function Selection

1. For regression challenges in CNN models, viable choices for the loss function include L1 loss or L2 loss.

2. When confronted with classification tasks, opt for a suitable loss function from the available options.

3. Cross entropy loss stands out as a prevalent choice, frequently employed in CNN models featuring a softmax layer at the conclusion.

4. Addressing concerns regarding compactness within a class or the margin between different classes necessitates considering enhancements based on cross entropy loss. Examples include center loss and large-margin softmax loss.

5. The choice of a loss function in CNNs should align with the specific application scenario. For instance, in face recognition applications, contrastive loss and triplet loss have emerged as commonly utilized options.

5.2.4. Optimizer

In CNNs, optimization of nonconvex functions is crucial, often necessitating optimizers to efficiently minimize the loss function within a reasonable timeframe [Li+21]. Nonconvex functions exhibit intricate landscapes with multiple local optima, making their optimization challenging. These functions lack the convex property, where any line segment connecting two points lies entirely within the function's domain, complicating the search for the global minimum.

Common optimization algorithms include momentum, Root-mean-square prop (RMSprop), adaptive moment estimation (Adam), etc.

Gradient Descent

Three gradient descent methods for training CNN models are batch gradient descent (BGD), stochastic gradient descent (SGD), and mini-batch gradient descent (MBGD).

- BGD is slow due to calculating the average gradient of the entire batch, making it impractical for large datasets or in-memory constraints.

- SGD, using one sample per update, is suitable for online learning but prone to high variance and oscillations.

- MBGD, a popular choice, combines advantages of BGD and SGD, efficiently using a small batch to stabilize convergence.

Gradient Descent Optimization Algorithms

Effective algorithms built on MBGD include:

- Momentum algorithm simulates physical momentum, preventing oscillations for faster convergence.

- Nesterov accelerated gradient (NAG) enhances momentum algorithm predictability to slow down before positive slopes.

- Adagrad adapts the learning rate to parameters, suitable for sparse data.

- Adadelta limits the monotonically decreasing learning rate of Adagrad.

- RMSprop addresses diminishing learning rate issues in Adagrad.

- Adam combines momentum with RMSprop, proving effective in various CNN structures.

- Other variants like AdaMax, Nadam, and AMSGrad offer improvements and constraints.

Guidelines for Optimizer Selection:

1. Utilize Mini-Batch Gradient Descent (MBGD) to strike a balance between Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD), considering both computing cost and the accuracy of each update.

2. Optimizer performance is intricately linked to data distribution. Adhering to the No Free Lunch Theorem, no single optimizer universally outperforms others in all scenarios. A prudent approach involves selecting optimizers based on their unique characteristics.

3. In instances of excessive oscillation or divergence, it is advisable to contemplate reducing the learning rate to address these issues.

5.3. Applications

The utilization of Convolutional Neural Networks (CNN) in computer vision has made possible achievements that were once deemed impossible over the past centuries. These accomplishments include facial recognition, autonomous vehicles, self-service supermarkets, and intelligent medical treatments [Li+21].

1-D CNN Applications

1. **Time Series Prediction:**
   - Applied to predict time series data, such as electrocardiogram (ECG) time series, weather forecast, and traffic flow prediction.
   - Example: Prediction of atrial fibrillation using short-term ECG data.
2. **Signal Identification:**
   - Used for discriminating input signals based on features learned from training data.
   - Applications include ECG signal identification, structural damage identification, and system fault identification.

2-D CNN Applications

keyword spotting: One of the key applications of artificial neural networks in the consumer domain is found in the realm of digital assistants, particularly in the area of automatic speech recognition. By employing automatic speech recognition, digital assistants empower users to effortlessly control devices through spoken commands. While automatic speech recognition has been in existence since the 1980s, utilizing methods such as Hidden Markov models and Gaussian mixture models, it was not until 2012 that artificial neural networks were introduced to address this challenge. This marked a significant enhancement in user experience and played a pivotal role in boosting the widespread adoption of digital assistants. Effectively managing audio data streams through continuous processing by an artificial neural network becomes challenging when dealing with multiple users. Additionally, the persistent recording of audio data raises valid privacy and security concerns. As a result, the device needs to identify a pre-defined keyword before transmitting audio data to the server, a procedure commonly known as keyword spotting [BC23].

Other 2-D applications of CNNs are:

1. **Image Classification:**
   - Task of classifying images into categories.

- Used in medical image classification, traffic scenes related classification, etc.
- Example: Custom CNN for the classification of interstitial lung disease.

2. **Object Detection:**
   - Involves identifying objects within images and marking them with bounding boxes.
   - Two-stage approaches like R-CNN, fast R-CNN, and faster R-CNN, as well as one-stage approaches like YOLO, are discussed.
   - Applications in detecting objects in images or videos.
3. **Image Segmentation:**
   - Divides an image into different areas and marks the boundaries of semantic entities.
   - Examples include U-Net for medical image segmentation and Panoptic segmentation.
   - Applications in medical image segmentation and semantic segmentation.
4. **Face Recognition:**
   - A biometric identification technique based on facial features.
   - Evolution from DeepFace to more recent approaches like FaceNet, VGGFace, etc.
   - Different loss functions, such as triplet loss, are employed for training.

Multidimensional CNN Applications

1. **Human Action Recognition:**
   - Involves recognizing human actions in videos.
   - 3D CNNs used to extract features, and integration with 2D CNN features is discussed.
2. **Object Recognition/Detection in 3D:**
   - Applications like object detection of RGBD images using 3D ShapeNet and VoxNet for 3D object recognition.
   - Detection of high-dimensional images like X-rays and CT images using 3D CNN.

5.4. Why CNN is relevant

Choosing a Convolutional Neural Network (CNN) for speech recognition is justified by the nature of the input data—spectrograms (see Section 5.7.2). Since spectrograms are two-dimensional arrays, a neural network must adeptly handle multidimensional tensors. CNNs, designed for precisely this purpose, excel in extracting features from relationships between adjacent values in such data [WS19].

CNNs, commonly associated with image processing, excel at learning hierarchical features, starting from simple elements like lines or edges and progressing to complex structures like eyes or ears. Importantly, their adaptability extends beyond images, making them well-suited for any multidimensional vector input.

Given that spectrogram data shares similarities with images as two-dimensional grids, CNNs are a natural fit. Their ability to capture relationships in a two-dimensional space aligns perfectly with the structure of spectrograms, making CNNs an effective choice for speech recognition, facilitating the extraction of meaningful features from the input data.

5.5. Hyperparameters

From tuning activation functions, loss functions, and optimizers to adjusting various other hyperparameters, the impact on model performance is substantial. It is well-established that there is no universally fixed set of hyperparameters that guarantees optimal solutions at all times. Consequently, relying on a combination of experience and established guidelines becomes crucial when navigating the intricate process of hyperparameter tuning [Li+21].

- **Learning Rate:** Refers to the step size of updating network weights. It can be constant or variable. It should be set in an appropriate range to balance convergence speed and stability.

- **Epoch:** The number of times the whole training set is input to the neural network for training. Should be adjusted based on the gap between training set and validation set accuracy to avoid underfitting or overfitting.

- **Mini-Batch Size:** The number of samples sent to the model in each training iteration. Too small or too large batch sizes can affect convergence and stability. Often set to dozens to hundreds, usually a power of 2 for Graphics Processing Unit (GPU) performance optimization.

- **Number of Conv Layers:** Determines the depth of the network and its ability to capture different-level features. More layers generally allow the representation of more complex features but can be harder to train.

- **Conv Kernel Size:** The size of convolution kernels in each layer. The smaller the kernel, the fewer parameters and computational complexity. Commonly used sizes include 3x3, 1x1, 1xn, and nx1.

- **Number of Filters (Kernels):** The number of filters in each convolutional layer, determining the depth and capacity of the network. Often increases in deeper layers.

- **Activation Function:** Introduces non-linearity to the network. Common choices include ReLU, Sigmoid, and Tanh. The choice depends on the nature of the problem and data characteristics.

5.6. Requirements

- **Model Size and Complexity:** Design a compact and computationally efficient CNN model that strikes a balance between accuracy and size, considering the limited memory and processing power of Arduino Nano.

- **Input Format:** Tailor the CNN to handle the specific format and characteristics of audio input suitable for Arduino Nano's limited input data capabilities.

- **Real-time Processing:** Optimize the CNN for real-time processing to ensure timely responses for speech recognition in the dynamic scenarios of Arduino Nano.

- **Energy Efficiency:** Develop an energy-efficient CNN that minimizes power consumption during inference without compromising performance, acknowledging the limited power resources on Arduino Nano.

- **Output Interface:** Align the CNN's output format with the device's capabilities to facilitate seamless integration with Arduino Nano, recognizing and utilizing its limited output capabilities.

- **Noise Robustness:** Incorporate features or preprocessing steps in the CNN to enhance noise robustness, adapting to different acoustic environments and addressing potential variability in environmental conditions.

- **Inference Speed:** Optimize the CNN for fast inference to ensure prompt recognition of spoken commands, considering the limited processing speed of Arduino Nano.
- **Training Considerations:** Design the CNN to be trainable with a manageable amount of data, considering the limited availability of training data for Arduino Nano, and address the specific nuances of the intended speech recognition application.
- **Postprocessing Techniques:** Implement postprocessing methods, such as score averaging or consensus-based recognition, to refine and enhance the accuracy of speech recognition in real-world scenarios with uncertainties.
- **Compatibility:** Ensure that the CNN model architecture and parameters are compatible with the hardware specifications of Arduino Nano, ensuring seamless integration and optimal performance on the device.

5.7. Input

In the realm of Convolutional Neural Networks (CNNs), understanding the intricacies of input data is paramount for effective model design and training. Here, the structure of input data is represented, emphasizing its role as a multi-dimensional array—commonly known as a tensor.

5.7.1. Input Data Specifications

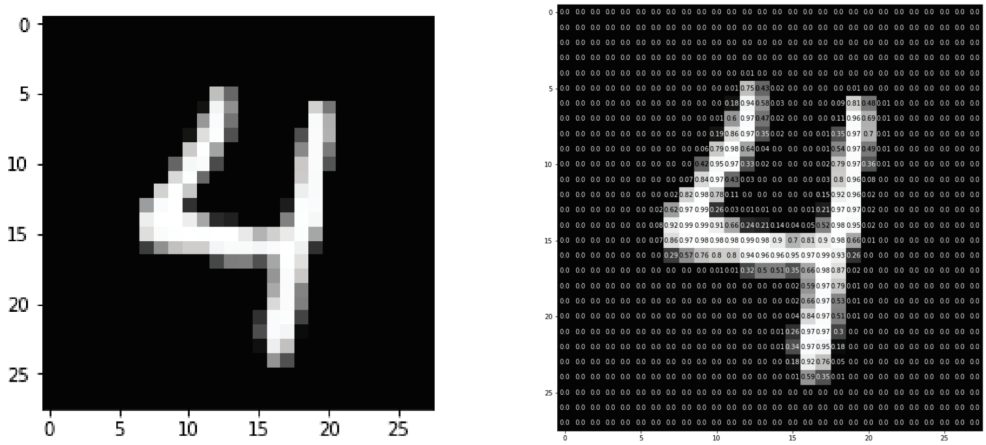
Dimensions for Image Data

The input data in a Convolutional Neural Network (CNN) is fundamentally structured as a multi-dimensional array, commonly referred to as a tensor. This tensor represents the raw input to the neural network and is pivotal for subsequent feature extraction and hierarchical representation learning.

Input Shape Specifications

For image data, the tensor dimensions typically encompass the height, width, and channels of the image. In the context of color images using the RGB color space, the shape of the tensor is commonly denoted as (height, width, 3). Here, "height" and "width" correspond to the spatial dimensions of the image, and "3" indicates the three color channels (Red, Green, Blue). For example, if you have an image with dimensions 100×100 pixels, the RGB representation would be a $100 \times 100 \times 3$ matrix, where each element corresponds to the intensity of the respective color channel at that pixel.

Grayscale images, on the other hand, have only one channel. Each pixel in a grayscale image is represented by a single value indicating the intensity of brightness. The range of intensity values depends on the bit depth; for an 8-bit image, values typically range from 0 (black) to 255 (white). The matrix representing a grayscale image would be a 2D matrix, where each element represents the intensity of a pixel. In the Figure 5.6 the digital presentation of the number "4" is presented. Note that the pixel values are normalized to be between 0 and 1, with 0 representing the black color and 1 representing the white color pixel value.



(a) (b)

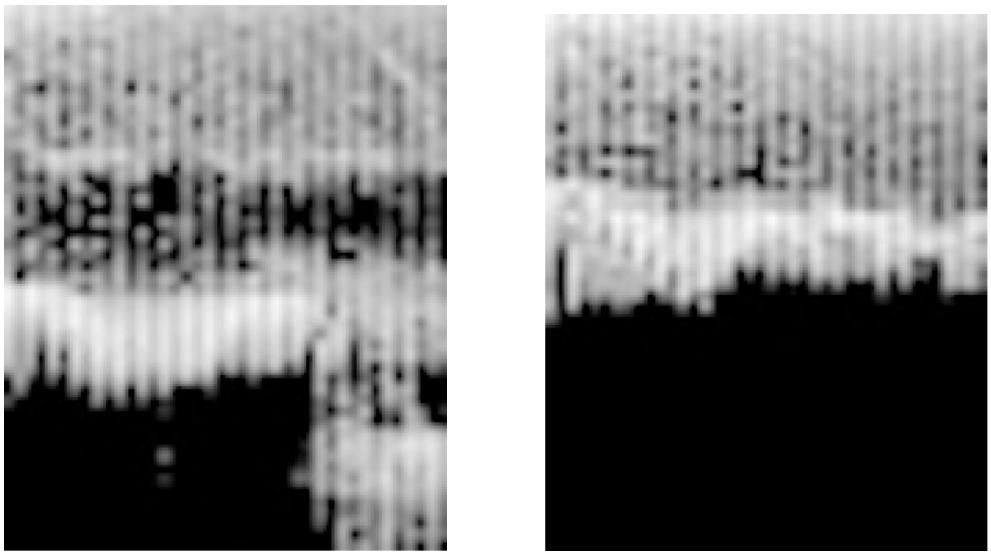
Figure 5.6.: Digital presentation of an image [SKP18]. (a) Digital presentation of the number "4." (b) Normalized pixel values of the number "4."

Normalization (Optional)

Normalization layers, such as Batch Normalization, can be optionally employed within the input layer to facilitate stable training. Normalization involves adjusting the input values to a layer, ensuring that they fall within a standardized range. This can enhance convergence during training and mitigate issues related to vanishing or exploding gradients.

5.7.2. Input Data for Speech Recognition Applications

It's important to note that he model does not process raw audio sample data; instead, it operates on spectrograms [WS19]; two dimensional arrays comprised of slices of frequency information, each derived from distinct time windows. Figure 5.7a visually depicts a spectrogram generated from a one-second audio clip of the word "yes," while Figure 5.7b illustrates the same for the word "no."



(a) (b)

Figure 5.7.: Spectral representations of "yes" and "no" [WS19]. (a) Spectral representation of the word "yes." (b) Spectral representation of the word "no."

By isolating frequency information during preprocessing, we simplify the model's training. It sidesteps the need to decipher raw audio data and instead operates with a higher-layer abstraction, distilling the most pertinent information. As a spectrogram is a two-dimensional array, it is inputted into the model as a 2D tensor.

Figure 5.7a depicts the input fed into the neural network, represented as a 2D array with a single channel, resembling a monochrome image. Working with 16 KHz audio sample data, the goal is to employ "feature generation" to transform a challenging input format (16,000 numerical values per second of audio) into a more machine-friendly representation. While machine vision often deals easily with images, domains like audio and natural language processing commonly require preprocessing before feeding data into a model.

To understand why preprocessing aids model comprehension, examine the original raw representations of audio recordings in Figures 5.8a through 5.8d. Without labels, distinguishing identical words is challenging. In contrast, Figures 5.9a through 5.9d demonstrate the spectrograms of the same recordings. Spectrograms, compared to raw waveforms, make differences more discernible, facilitating model interpretation.

Additionally, generated spectrograms are much smaller than raw sample data, with each containing 1,960 numeric values instead of 16,000 in the waveform. This summarization reduces the neural network's workload. While models like DeepMind's WaveNet [Oor+16] can handle raw data,

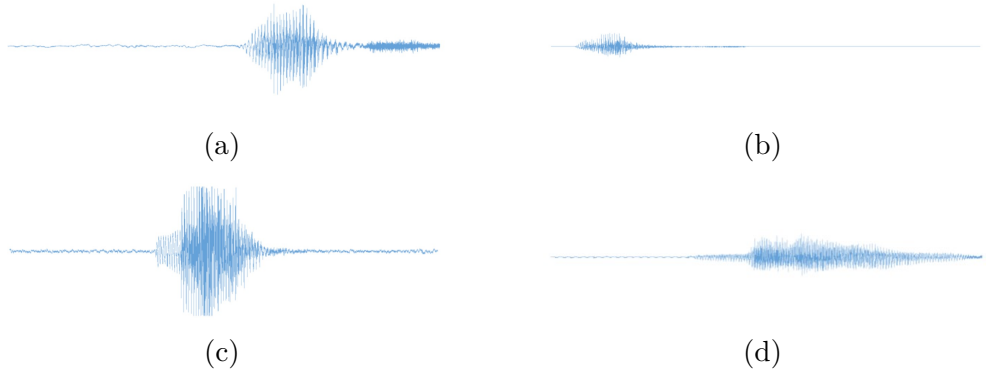


Figure 5.8.: Audio waveforms of "yes" and "no" [WS19]. (a) "yes" audio waveform. (b) Another "yes" audio waveform. (c) "no" audio waveform. (d) Another "no" audio waveform.

they often entail more computation than the combination of a neural network with hand-engineered features, making the latter preferable for resource-constrained environments like embedded systems.

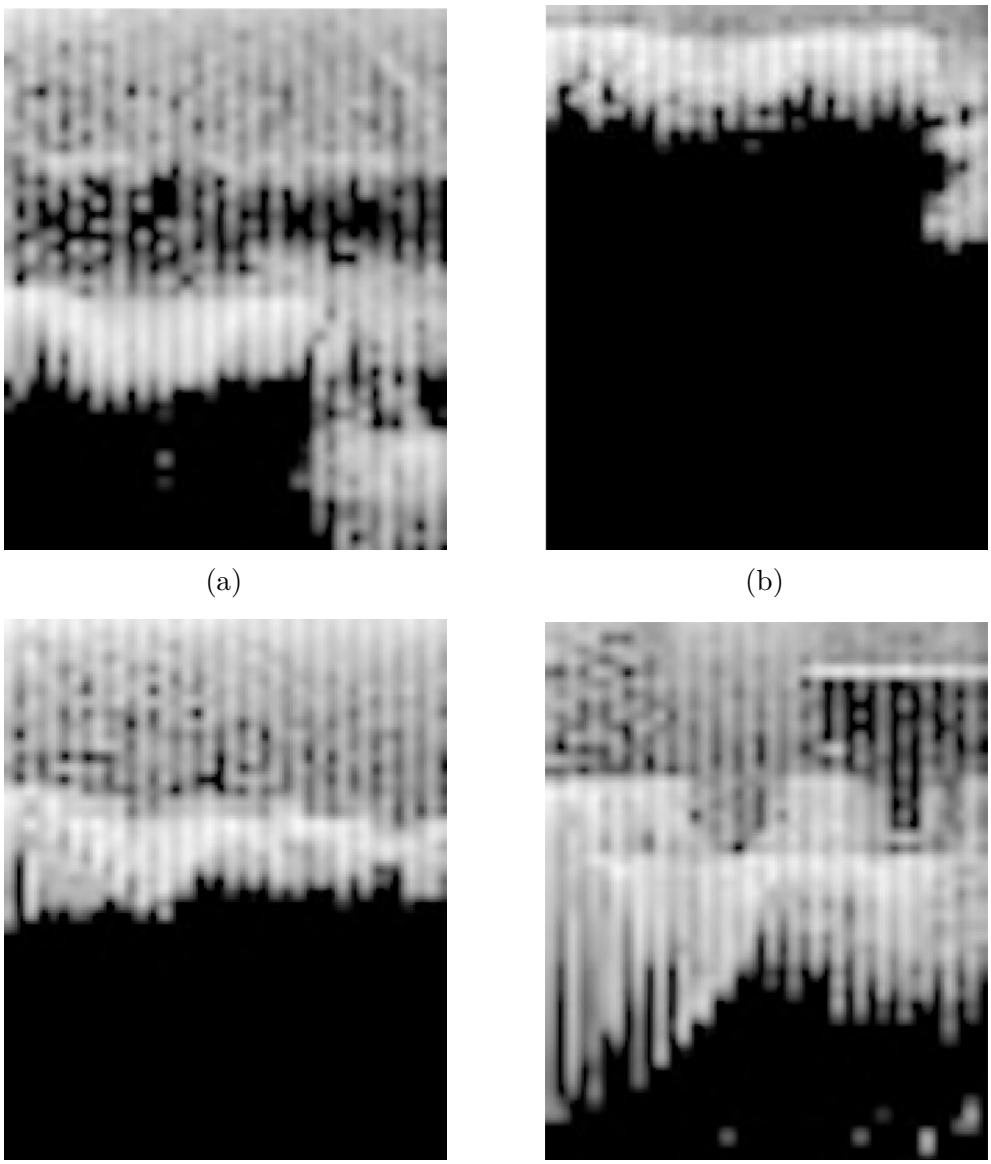
5.7.3. How to Convert Audio to a Spectrogram

The process of converting audio into a spectrogram involves creating a 2D array to represent an audio snippet, with each row reflecting a segment of the audio split into frequency buckets. This occurs in a loop, generating new features efficiently for the time elapsed since the last iteration [WS19].

For each row, a segment of the audio undergoes fast Fourier transform (FFT) to analyze frequency distribution, resulting in condensed frequency information. The 2D array is formed by combining results from consecutive slices of the audio signal, ensuring continuity.

The decision on which slices to generate is based on time intervals between operations, optimizing computational efficiency. The loop retrieves audio samples for each new slice, and after processing, a comprehensive spectrogram is obtained. The subsequent steps involve using this spectrogram for inference with the model, and the results are interpreted for recognition.

In the Figure 5.10 the diagram of audio samples being processed is shown. The process in this example involves converting audio into a spectrogram by analyzing one-second audio snippets in a loop. Each 30-millisecond audio segment undergoes a fast Fourier transform (FFT), producing an array of 256 frequency buckets, which are then averaged into 43 buckets to distill relevant frequency information. These steps are repeated for consecutive segments with a 20-millisecond overlap, ensuring continuity. The resulting frequency data forms a 2D array, representing the entire one-second audio sample.



(a) (b)

(c) (d)

Figure 5.9.: Audio Spectrograms of "yes" and "no" [WS19]. (a) "yes" audio spectrogram (b) Another "yes" audio spectrogram. (c) "no" audio spectrogram. (d) Another "no" audio spectrogram.

5.8. Output

In this section, the output of the CNN algorithm is explained. Initially, the output of the CNN for classification and regression tasks are explained in general terms. Subsequently, the output of the model for speech recognition applications is explained.

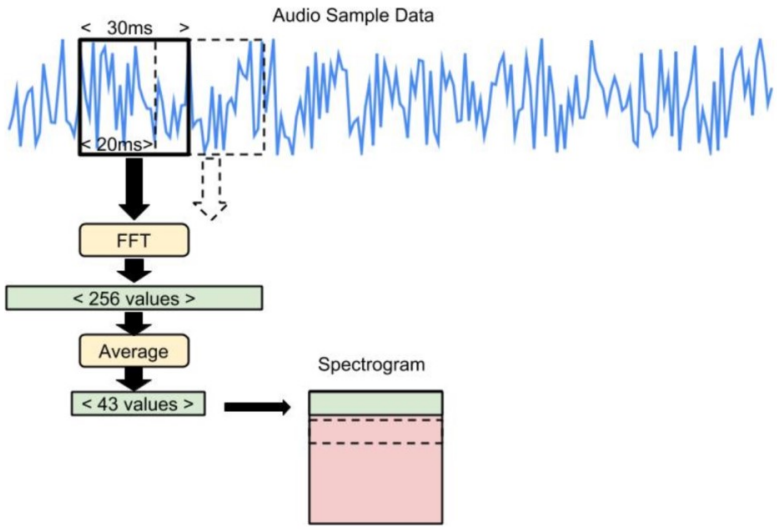


Figure 5.10.: Diagram of the processing of audio samples [WS19].

5.8.1. The Output of the CNN Algorithm for Classification and Regression Tasks

Classification

In a classification task, a Convolutional Neural Network (CNN) produces an output that represents a probability distribution over different classes. This is achieved through the use of a softmax activation function (explained later) in the final layer, ensuring that the output values signify probabilities. The class with the highest probability is then considered the predicted class.

Regression

In a regression task, the CNN output is a continuous numerical value. The network is designed to predict a specific numerical outcome, and the final layer often uses an activation function, such as linear, that allows for unbounded output. The training process involves minimizing the difference between the predicted value and the actual target value, as determined by the chosen loss function.

5.8.2. Output of the Model for Speech Recognition Applications

The CNN model in the case of speech recognition operates as a classifier, generating class probabilities as output. The final outcome of the model is determined by the output of the softmax (explained later) layer, resulting in, for example, four numbers corresponding to categories: "silence,"

"unknown," "yes," and "no" [WS19]. These numerical values represent the scores for each category, and the category with the highest score is the model's prediction, indicating the model's confidence in that prediction. For instance, if the model outputs `[10, 4, 231, 80]`, it predicts that the third category, "yes," is the most likely result with a score of 231.

To avoid word recognition failure the model needs to run more frequently than once per second. Postprocessing methodologies are implemented to refine the model's predictions over time. A common approach involves averaging scores obtained from multiple runs, contributing to a more stable and reliable output. Recognition events are often triggered when there is a consensus of high scores for the same word within a condensed timeframe. This adaptive strategy enhances the robustness of the speech recognition system, allowing it to adapt to variations in speech patterns and environmental conditions.

Upon successful recognition, the command responder utilizes the device's output capabilities, such as flashing an LED or displaying information on an LCD screen. The model's output is structured with two dimensions, where the first is a wrapper and the second holds the probabilities for each class. This output format aligns with the requirements for our embedded hardware implementation on Arduino Nano 33 BLE Sense.

The Softmax Function

The softmax function is commonly used in Convolutional Neural Networks (CNNs) and other machine learning models, particularly in the output layer, to convert a vector of raw scores or logits into probabilities. It essentially normalizes the input values into a probability distribution that sums to 1 [SKP18].

In the context of CNNs, the softmax function is often applied to the output layer when the network is used for classification tasks. Here's the formula for the softmax function:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{5.5}$$

Wwhere x_i is the raw score or logit for class i, K is the total number of classes, and e is the base of the natural logarithm (Euler's number). An exmple of applying the Softmax function is shown in the Figure 5.11.

5.9. Python Example Code

The identification of regional patterns within an image is facilitated by the convolutional layer. Following the convolutional layer, the max pooling

Figure 5.11.: Example of applying the softmax function [SKP18]

layer is employed to diminish dimensionality. This section illustrates image classification by using a code provided by Sewak et al [SKP18].

It is crucial to initially standardize all images to a uniform size. The initial convolution layer necessitates an additional parameter, `input.shape()`. The focus here is on training a Convolutional Neural Network (CNN) for image classification using the CIFAR-10 database. CIFAR-10 comprises 60,000 color images, each of size 32×32. These images are categorized into 10 classes, with 6,000 images per category, namely airplane, automobile, bird, cat, dog, deer, frog, horse, ship, and truck.

Imports

In the Listing 5.1 are the necessary libraries and modules required for working with neural networks, image data, and visualization. The versions are shown in the Table 5.1.

- Python version: 3.9.

Table 5.1.: Versions of Libraries

Library	Version
Numpy	1.23.5
Matplotlib	3.7.1
Keras	2.12.0

Load CIFAR-10 Dataset

In Listing 5.2 the CIFAR-10 dataset is loaded.CIFAR-10 is a dataset of 50,000 32×32 color training images and 10,000 test images.

Data Preprocessing

In the Listing 5.3 the image data is normalized and one-hot encoding is performed on the labels. The training set is split into training and validation sets.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
    , Dropout
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

Code/CNN/CNNDataMining.py

Listing 5.1.: Importing necessary libraries and modules.

```
(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()
```

Code/CNN/CNNDataMining.py

Listing 5.2.: Loading and preparing the CIFAR-10 dataset.

```
# rescale [0,255] -> [0,1]
xTrain = xTrain.astype('float32') / 255

# one-hot encode the labels
numClasses = len(np.unique(yTrain))
yTrain = np_utils.to_categorical(yTrain, numClasses)
yTest = np_utils.to_categorical(yTest, numClasses)

(xTrain, xValid) = xTrain[5000:], xTrain[:5000]
(yTrain, yValid) = yTrain[5000:], yTrain[:5000]
```

Code/CNN/CNNDataMining.py

Listing 5.3.: Preprocessing data: normalization, one-hot encoding, and splitting into training and validation sets

Augmented Image Generator

In the Listing 5.4 image data generators for training and validation is created and configured. These generators will perform data augmentation, such as shifting and flipping, to increase the diversity of the training set.

Plot the First Nine Images of Cifar-10

Listing 5.5 loads the cifar-10 dataset and plots the first nine images as shown in Figure 5.12.

```
datagenTrain = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

datagenValid = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# Fitting augmented image generator on data
datagenTrain.fit(xTrain)
datagenValid.fit(xValid)
```

Code/CNN/CNNDataMining.py

Listing 5.4.: Configuring image data generators for augmentation and fitting them on training and validation data

```
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(xTrain[i])
plt.tight_layout()  # Adjust layout for better visualization
plt.savefig('CIFAR10FirstNineImages.png')  # Save the figure
plt.show()
```

Code/CNN/CNNDataMining.py

Listing 5.5.: Loading and visualizing the first nine images from the CIFAR-10 dataset

CNN Model Definition

A Convolutional Neural Network (CNN) model is defined in Listing 5.6 using Keras with convolutional layers, max pooling, dropout for regularization, and dense layers.

Compile the Model

In Listing 5.7 the model is compiled with the specified loss function, optimizer, and evaluation metric.

Train the Model with Augmented Data

In Listing 5.8 the model is trained using the augmented data generators. This involves calling the `fit_generator` function instead of `fit` and

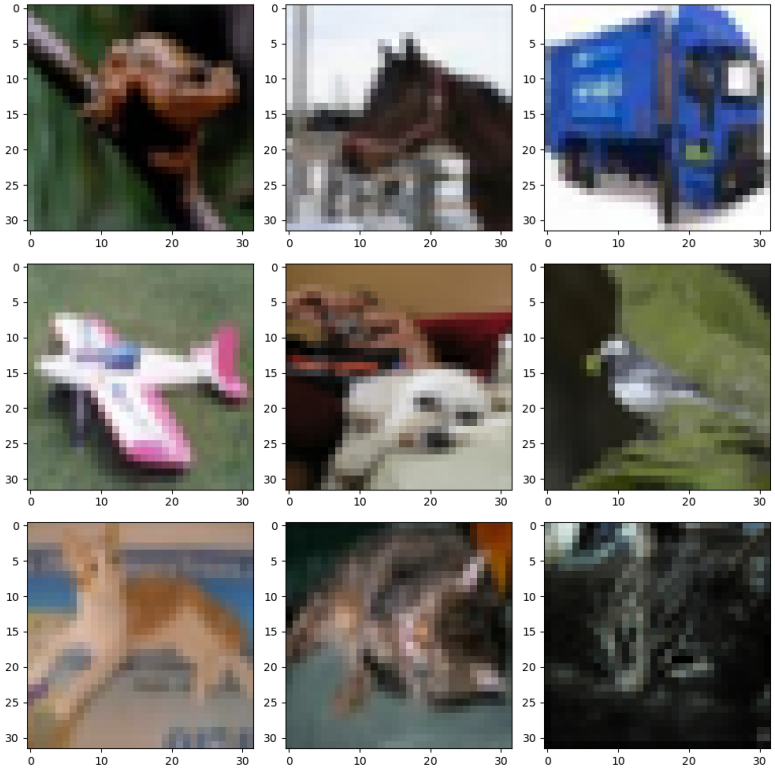


Figure 5.12.: First nine images from the CIFAR-10 dataset.

```
model.add(Conv2D(filters=32, kernel_size=2, padding='same',
    activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

Code/CNN/CNNDataMining.py

Listing 5.6.: Defining a Convolutional Neural Network (CNN) model using Keras

providing the data generators for training and validation sets.

Plotting the Loss and Accuracy Curves

In Listing 5.9 the accuracy and loss curves are plotted. The results are shown in the Figure 5.13. The observed trends in the training and validation metrics, with a downward trend in both training and validation

```
model.compile(loss='categorical_crossentropy', optimizer='
    rmsprop', metrics=['accuracy'])
```

Code/CNN/CNNDataMining.py

Listing 5.7.: Compiling the CNN model with specified loss function, optimizer, and metric

```
hist = model.fit_generator(
    datagenTrain.flow(xTrain, yTrain, batch_size=32),
    steps_per_epoch=len(xTrain) // 32,
    epochs=10,
    validation_data=datagenValid.flow(xValid, yValid,
    batch_size=32),
    validation_steps=len(xValid) // 32,
    callbacks=[checkpointer],
    verbose=2,
    shuffle=True
)
```

Code/CNN/CNNDataMining.py

Listing 5.8.: Training the CNN model with augmented data using data generators

loss and an upward trend in accuracy, indicate positive learning and generalization behavior of the neural network. The somewhat unconventional scenario of validation loss being below training loss and validation accuracy exceeding training accuracy could be influenced by effective data augmentation, contributing to the model's robustness. These trends collectively suggest that the model is not overfitting the training data and is likely to generalize well to unseen data, highlighting successful training and potential for further improvement.

5.10. Conclusion

The Convolutional Neural Network (CNN) algorithm stands out as a powerful and versatile tool in the field of image processing and pattern recognition. Through its unique architecture, which includes convolutional layers for feature extraction and pooling layers for spatial down-sampling, CNNs have demonstrated remarkable success in tasks such as image classification, object detection, and facial recognition.

The effectiveness of a CNN is intricately tied to the quality and representativeness of its training data. If the dataset used for training is flawed, biased, or insufficient, the network is likely to learn and perpetuate those

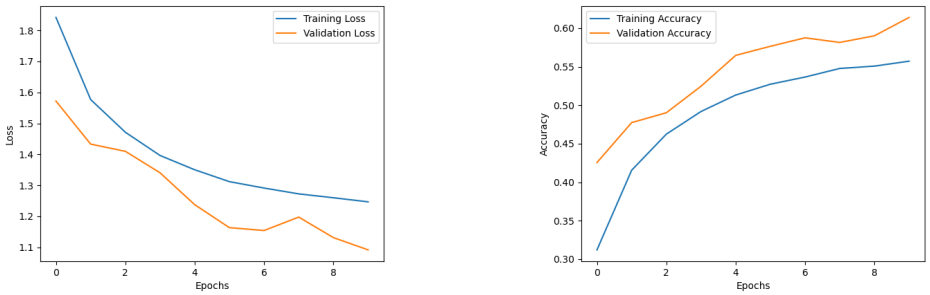
```
trainingLoss = hist.history['loss']
trainingAccuracy = hist.history['accuracy']
validationLoss = hist.history['val_loss']
validationAccuracy = hist.history['val_accuracy']

# Plotting and saving the loss curve
plt.plot(trainingLoss, label='Training Loss')
plt.plot(validationLoss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('lossPlot.png')
plt.show()

# Plotting and saving the accuracy curve
plt.plot(trainingAccuracy, label='Training Accuracy')
plt.plot(validationAccuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('accuracyPlot.png')
plt.show()
```

Code/CNN/CNNDataMining.py

Listing 5.9.: Plotting the loss and accuracy curves



(a) (b)

Figure 5.13.: (a) Training and validation loss trends over epochs. (b) Training and validation accuracy trends over epochs.

shortcomings, leading to suboptimal performance in real-world applications. This principle underscores the critical importance of meticulous data curation, ensuring that the input data is not only abundant but also diverse, accurate, and devoid of biases. Taking into account domain knowledge is crucial to avoid the "garbage in, garbage out" pitfall when engaged in the application of machine learning algorithms, as depicted in the Figure 5.14.

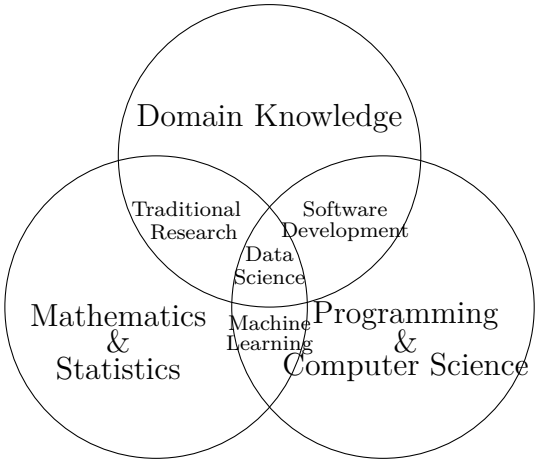


Figure 5.14.: Key areas of expertise, data science venn diagram

Part III.

Important Python Packages

6. TensorFlow

6.1. Introduction

In the ever-evolving landscape of machine learning and artificial intelligence, TensorFlow stands as a cornerstone library, empowering developers and researchers with a robust platform for numerical computations. Originally developed by the Google Brain team, TensorFlow has become a driving force behind large-scale services at Google, including Google Cloud Speech, Google Photos, and Google Search. Since its open-source debut in November 2015, TensorFlow has grown to be the most widely adopted deep learning library in the industry.

TensorFlow's appeal lies in its ability to seamlessly blend flexibility and efficiency, offering a versatile toolkit for a myriad of machine learning applications. From image classification and natural language processing to recommender systems and time series forecasting, TensorFlow provides a comprehensive framework that scales from research experiments to real-world applications.

It enables the training and execution of very large neural networks efficiently by distributing computations across potentially hundreds of multi-GPU servers [Gér22]:

- **Origin:** TensorFlow was created at Google and supports many of its large-scale machine learning applications.
- **Open Source:** TensorFlow was open-sourced in November 2015.
- **Version:** Version 2.0 was released in September 2019.
- **Integration with Keras:** TensorFlow comes bundled with Keras, and Keras relies on TensorFlow for all intensive computations.

6.1.1. TensorFlow Framework

TensorFlow Overview

TensorFlow serves as a scalable machine learning system operating in diverse environments. Employing dataflow graphs to represent computation, shared state, and state-altering operations, TensorFlow is proficient in training and executing deep neural networks for various applications,

including handwritten digit classification, image recognition, word embeddings, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and PDE-based simulations. The framework efficiently maps dataflow graph nodes across multiple machines in a cluster and across various computational devices within a machine, such as multicore CPUs, general-purpose GPUs, and specialized ASICs known as Tensor Processing Units (TPUs). TensorFlow facilitates experimentation with novel optimizations and training algorithms, with a primary focus on deep neural network training and inference. Widely used in Google services, TensorFlow is released as an open-source project and has become a staple in machine learning research, showcasing compelling performance across real-world applications.

TensorFlow Use Case

TensorFlow, developed by Google, is a versatile framework for creating machine learning applications and training models. Google's search engine itself relies on TensorFlow for AI applications. For instance, when a user types a query into the Google search bar, the search engine predicts the most suitable word completion based on previous searches. TensorFlow is applicable across various tasks, with a particular emphasis on training and inferring deep neural networks. It offers diverse workflows for model development and training using programming languages like Python and JavaScript. After training, models can be deployed on the cloud or any device for edge computing applications, regardless of the language used on the device. The crucial aspect of every machine learning application is training the model to make real-time decisions without human intervention.

TensorFlow Lite for Microcontrollers

TensorFlow is a crucial piece of infrastructure for training machine learning models and teaching algorithms to identify patterns in data. However, when deploying on small embedded microcontrollers, the need arises to streamline TensorFlow for efficiency. TensorFlow Lite, or TFLite, addresses this need by providing tools to convert and optimize TensorFlow models for mobile and edge devices. It is designed to be lean and mean, catering to the requirements of resource-constrained edge devices. TensorFlow Lite for Microcontrollers is specifically tailored for running on devices with limited memory, enabling the enhancement of intelligence in billions of devices, including household appliances and Internet of Things (IoT) devices. This implementation brings machine learning to tiny microcontrollers, reducing dependence on expensive hardware or reliable internet connections and ensuring data security by keeping data on the device. TensorFlow Lite excels in on-device machine learning (Edge ML)

and is optimized for deployment to resource-constrained edge devices [DMM20].

6.2. Description

6.2.1. Overview of TensorFlow

TensorFlow stands out as a potent library for numerical computations [Gér22], especially tailored for extensive machine learning applications (though it can be employed for any task demanding substantial computations). Developed by the Google Brain team, TensorFlow drives various large-scale services at Google, including Google Cloud Speech, Google Photos, and Google Search. Open-sourced in November 2015, it has since become the most widely utilized deep learning library in the industry. Countless projects leverage TensorFlow for diverse machine learning tasks, such as image classification, natural language processing, recommender systems, and time series forecasting.

So, what does TensorFlow provide? Here is a summary:

- Its core closely resembles NumPy but with GPU support.
- It supports distributed computing across multiple devices and servers.
- It incorporates a just-in-time (JIT) compiler optimizing computations for speed and memory usage by extracting and optimizing the computation graph.
- Computation graphs can be exported to a portable format, allowing training in one environment and running in another.
- It implements reverse-mode autodiff and offers excellent optimizers like RMSProp and Nadam.

TensorFlow offers numerous features built upon these core capabilities, with Keras being the most notable. Additionally, it includes data loading and preprocessing operations (tf.data, tf.io, etc.), image processing operations (tf.image), signal processing operations (tf.signal), and more.

At the lowest level, each TensorFlow operation (op) is implemented using highly efficient C++ code, with multiple implementations called kernels dedicated to specific device types like CPUs, GPUs, or TPUs (tensor processing units). GPUs and TPUs significantly accelerate computations.

TensorFlow's architecture, depicted in Figure 6.1, illustrates that while high-level APIs like Keras and tf.data are often used, the lower-level Python API allows direct handling of tensors. TensorFlow's execution

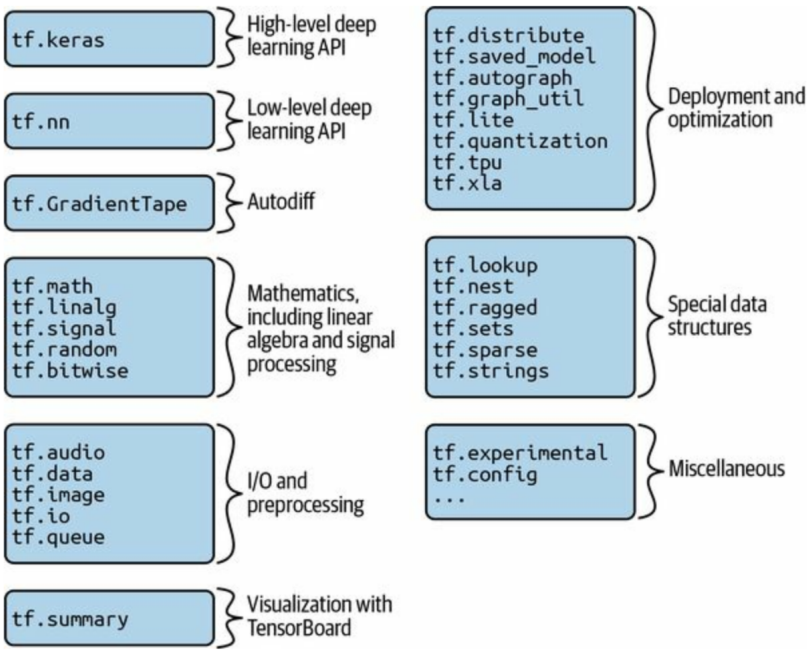


Figure 6.1.: TensorFlow's Python API [Gér22]

engine efficiently manages operations, even across multiple devices and machines if specified.

TensorFlow runs on Windows, Linux, macOS, and mobile devices through TensorFlow Lite, supporting both iOS and Android. APIs for other languages, including C++, Java, Swift, and TensorFlow.js (JavaScript implementation), are also available.

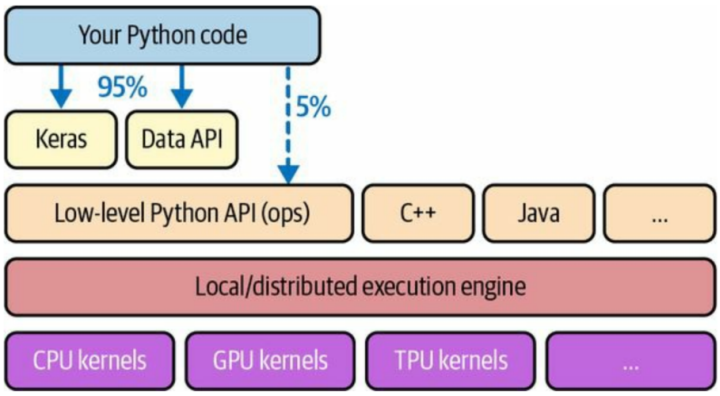


Figure 6.2.: TensorFlow's architecture [Gér22]

Beyond being a library, TensorFlow anchors an extensive ecosystem of libraries. TensorBoard aids visualization, and TensorFlow Extended (TFX) by Google helps in the productionization of TensorFlow projects, offering tools for data validation, preprocessing, model analysis, and

serving. TensorFlow Hub facilitates easy downloading and reuse of pretrained neural networks. The TensorFlow model garden provides various neural network architectures, some pretrained. Explore TensorFlow Resources, `https://github.com/jtoy/awesome-tensorflow`, for more TensorFlow-based projects, and find numerous projects on GitHub.

6.2.2. Data Structures Supported

TensorFlow supports many data structures [Gér22]:

- **Tensors (`tf.Tensor`):** The essence of TensorFlow's API centers on tensors, which traverse from one operation to another, hence the name TensorFlow. A tensor closely resembles a NumPy ndarray, typically representing a multidimensional array but capable of containing a scalar, like a simple value such as 42. The significance of these tensors becomes evident as we delve into creating custom cost functions, metrics, layers, and additional functionalities.

- **Variables (`tf.Variable`):** A tf.Variable behaves akin to a tf.Tensor: you can carry out similar operations, seamlessly integrate it with NumPy, and it exhibits the same strictness with types. However, it distinguishes itself by being mutable, allowing modifications in place.

- **Sparse Tensors (`tf.SparseTensor`):** Efficiently represent tensors containing mostly zeros. The tf.sparse package contains operations for sparse tensors.

- **Tensor Arrays (`tf.TensorArray`):** Lists of tensors with a default fixed length, but can optionally be made extensible. All tensors they contain must have the same shape and data type.

- **Ragged Tensors (`tf.RaggedTensor`):** Represent lists of tensors with the same rank and data type, but varying sizes. The dimensions along which the tensor sizes vary are called the ragged dimensions. The tf.ragged package contains operations for ragged tensors.

- **String Tensors:** Regular tensors of type `tf.string` representing byte strings, not Unicode strings. Unicode strings can be represented using tensors of type tf.int32. The tf.strings package contains ops for byte strings and Unicode strings.

- **Sets:** Are represented as regular tensors (or sparse tensors). Each set is represented by a vector in the tensor's last axis. You can manipulate sets using operations from the `tf.sets` package.

- **Queues:** Store tensors across multiple steps. TensorFlow offers various kinds of queues: basic first-in, first-out (FIFO) queues (FIFOQueue), plus queues that can prioritize some items (PriorityQueue), shuffle their items (RandomShuffleQueue), and batch items of different shapes by padding (PaddingFIFOQueue). These classes are all in the `tf.queue` package.

6.2.3. Functions and Graphs in TensorFlow

In the earlier days of TensorFlow 1, working with graphs was unavoidable due to their central role in TensorFlow's API, albeit with inherent complexities. With the release of TensorFlow 2 in 2019, while graphs are still present, they are no longer as central and have become considerably simpler to utilize. To illustrate this simplicity, let's consider a basic function that calculates the cube of its input [Gér22]:

```
def cube(x):
  return x ** 3
```

Listing 6.1: Python function to calculate the cube

This function can be called with a Python value like an integer or a float, as well as with a tensor:

```
cube(2)
```

Listing 6.2: Calling the Python function with a Python value

```
8
```

```
cube(tf.constant(2.0))
```

Listing 6.3: Calling the Python function with a tensor

```
<tf.Tensor: shape=(), dtype=float32, numpy=8.0>
```

Now, let's use `tf.function()` to convert this Python function into a TensorFlow function:

```
tf_cube = tf.function(cube)
tf_cube
```

Listing 6.4: Converting Python function to TensorFlow function

```
<tensorflow.python.eager.def_function.Function at 0x7fbfe0c54d50>
```

This TensorFlow function can be used just like the original Python function, yielding the same results (always as tensors):

```
tf_cube(2)
```

Listing 6.5: Using the TensorFlow function

```
<tf.Tensor: shape=(), dtype=int32, numpy=8>
```

```
tf_cube(tf.constant(2.0))
```

Listing 6.6: Using the TensorFlow function with a tensor

```
<tf.Tensor: shape=(), dtype=float32, numpy=8.0>
```

Beneath the surface, `tf.function()` analyzes the computations performed by the `cube()` function and generates an equivalent computation graph. As demonstrated, this process is relatively straightforward.

Alternatively, we could have used `tf.function` as a decorator, which is a more common practice:

```
@tf.function
def tf_cube(x):
  return x ** 3
```

Listing 6.7: Using tf.function as a decorator

The original Python function is still accessible through the TF function's `python_function` attribute when needed:

```
tf_cube.python_function(2)
```

Listing 6.8: Accessing the original Python function

```
8
```

TensorFlow optimizes the computation graph by pruning unused nodes, simplifying expressions (e.g., replacing $1 + 2$ with 3), and more. Once the optimized graph is ready, the TF function efficiently executes the operations in the graph, in the appropriate order (and in parallel when possible). Consequently, a TF function generally runs much faster than the original Python function, especially for complex computations. In most cases, transforming a Python function into a TF function is all you need to boost its performance.

6.3. TensorFlow Installation

To install TensorFlow, follow the step-by-step instructions outlined below. For additional details and updates, refer to the official TensorFlow installation guide.

6.3.1. Hardware Requirements

Note: TensorFlow binaries utilize AVX instructions, which may not be compatible with older CPUs.

The following devices with GPU support are compatible:

- NVIDIA® GPU card with CUDA® architectures 3.5, 5.0, 6.0, 7.0, 7.5, 8.0, and higher (Refer to the list of CUDA®-enabled GPU cards).
- For GPUs with unsupported CUDA® architectures or to circumvent JIT compilation from PTX, or to use different versions of the NVIDIA® libraries, refer to the Linux build-from-source guide.
- TensorFlow packages exclude PTX code except for the latest supported CUDA® architecture. Consequently, TensorFlow fails to load on older GPUs when CUDA_FORCE_PTX_JIT=1 is set (See Application Compatibility for details).

6.3.2. System Requirements

- Ubuntu 16.04 or later (64-bit)
- macOS 10.12.6 (Sierra) or later (64-bit) without GPU support
- Windows Native - Windows 7 or later (64-bit) without GPU support after TF 2.10
- Windows WSL2 - Windows 10 version 19044 or later (64-bit)
  Note: GPU support is available for Ubuntu and Windows with CUDA®-enabled cards.

6.3.3. Software Requirements and Dependencies

- Python version 3.9–3.11
- For Linux (requiring manylinux2014 support) and Windows, pip version 19.0 or higher is needed. For macOS, pip version 20.3 or higher is required.
- Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, and 2019 is essential for Windows Native installations.

The following NVIDIA® software is mandatory for GPU support:

- NVIDIA® GPU drivers version 450.80.02 or higher.

- CUDA® Toolkit 11.8.
- cuDNN SDK 8.6.0.
- (Optional) TensorRT can be installed to enhance latency and throughput for inference.

6.3.4. Step-by-step Installation Instructions

Windows Native

Note: TensorFlow 2.10 was the last release that supported GPU on native Windows. Starting from TensorFlow 2.11, consider installing TensorFlow in WSL2 or use `tensorflow-cpu`. Optionally, explore TensorFlow-DirectML-Plugin.

1. **System Requirements**

   Ensure Windows 7 or higher (64-bit).

   Note: Starting with TensorFlow 2.10, Windows CPU builds for x86/x64 processors are provided by Intel. Installing `tensorflow` or `tensorflow-cpu` installs Intel's `tensorflow-intel` package. These packages are provided as-is, and TensorFlow will make efforts to maintain their availability and integrity. Refer to a blog post for more on this collaboration.

2. **Install Microsoft Visual C++ Redistributable**

   Install Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, and 2019. For TensorFlow 2.1.0 and above, `msvcp140_-1.dll` from this package is required. Download it separately if not included with Visual Studio 2019.

   a) Go to Microsoft Visual C++ downloads.

   b) Navigate to Visual Studio 2015, 2017, and 2019 section.

   c) Download and install Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, and 2019 according to your platform.

   Ensure long paths are enabled on Windows.

3. **Install Miniconda**

   Miniconda is the recommended method for installing GPU-supported TensorFlow. It creates a dedicated environment, avoiding changes to your system's installed software, making GPU setup easier.

   Download the Miniconda Windows Installer, run it, and follow on-screen instructions.

4. **Create a Conda Environment**

   Create a new Conda environment named `tf` with the following command.

   ```
   conda create --name tf python=3.9
   ```

   Deactivate and activate it with the following commands.

   ```
   conda deactivate
   conda activate tf
   ```

   Ensure it stays activated for the rest of the installation.

5. **GPU Setup**

   Skip this section if running TensorFlow only on CPU.

   First, install NVIDIA GPU driver if not already done.

   Then, install CUDA and cuDNN with Conda.

   ```
   conda install -c conda-forge cudatoolkit=11.2 cudnn=8.1.0
   ```

6. **Install TensorFlow**

   TensorFlow requires an up-to-date pip installation. Upgrade pip to the latest version.

   ```
   pip install --upgrade pip
   ```

   Install TensorFlow using pip.

   Note: Avoid using Conda for TensorFlow installation as it may not have the latest stable version. Pip is recommended since TensorFlow is officially released on PyPI.

   ```
   # Versions above 2.10 are not supported on GPU on Windows Native
   pip install "tensorflow<2.11"
   ```

7. **Verify the Installation**

   Verify the CPU setup:

   ```
   python -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.norm
   ```

   If a tensor is returned, TensorFlow is successfully installed.

   Verify the GPU setup:

   ```
   python -c "import tensorflow as tf; print(tf.config.list_physical_devi
   ```

   If a list of GPU devices is returned, TensorFlow is successfully installed.

MacOS

1. **System Requirements:**

   Ensure your MacOS version is 10.12.6 (Sierra) or higher (64-bit).

   Note: TensorFlow supports Apple Silicon (M1), but packages with custom C++ extensions for TensorFlow must be compiled for Apple M1. While some packages like `tensorflow_decision_forests` offer M1-compatible versions, others may not. For those, use TensorFlow with x86 emulation and Rosetta. GPU support for TensorFlow on MacOS is not officially available, and these instructions are intended for CPU usage.

2. **Check Python Version:**

   Verify that your Python environment is configured correctly.

   Note: This process requires Python 3.9–3.11 and `pip >= 20.3` for MacOS.

   ```
   python3 --version
   python3 -m pip --version
   ```

3. **Install TensorFlow:**

   Upgrade your pip installation to the latest version before installing TensorFlow.

   ```
   pip install --upgrade pip
   ```

   Now, install TensorFlow using pip.

   ```
   pip install tensorflow
   ```

4. **Verify the Installation:**

   Confirm the successful installation of TensorFlow by running the following command:

   ```
   python3 -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.nor
   ```

   If a tensor is returned, TensorFlow has been installed successfully.

6.4. Example - Manual

Installation and Setup

1. Download and install Python: Visit the official Python website (`https://www.python.org`) and download the latest version of Python for your operating system. Follow the installation instructions provided.

2. Install PyCharm: Visit the JetBrains website (`https://www.jetbrains.com/pycharm`) and download PyCharm Community Edition, which is the free version. Install PyCharm by following the installation instructions specific to your operating system.

3. Install required Python packages:
   - TensorFlow 2.15.0: Open a command prompt or terminal and run the following command:

     ```
     pip install tensorflow==2.15.0
     ```

   - NumPy 1.26.3: Run the following command:

     ```
     pip install numpy==1.26.3
     ```

   - Matplotlib 3.8.0: Execute the following command:

     ```
     pip install matplotlib==3.8.0
     ```

6.4.1. User Manual for Running the Python Example File in PyCharm

1. Launch PyCharm: Open the PyCharm application from your desktop or applications menu.

2. Create a new project: Click on `Create New Project` or go to `File` → `New Project`. Choose a suitable location for your project and provide a name.

3. Open the Python file: Once the project is created, navigate to the project directory in the PyCharm project view. Right-click on the desired folder and select `New` → `Python File`. Provide a name for the file and click `OK`.

4. Copy your Python code: Open your Python file (with .py extension) in a text editor and copy the contents.

5. Paste the code: Paste the copied code into the newly created Python file in PyCharm.

6.4.2. Running the Script in Command Line

To run the Python script `clothingimgClassification.py` from the command line, follow these steps:

1. Open a command prompt or terminal window.

2. Navigate to the directory where the `clothingimgClassification.py` file is located using the `cd` command. For example:

```
cd path/to/script/directory
```

3. Once you are in the correct directory, run the script using the following command:

   ```
   python clothingimgClassification.py
   ```

   Note: If your system uses Python 3, you might need to use `python3` instead:

   ```
   python3 clothingimgClassification.py
   ```

   Replace `python` with the appropriate command based on your Python installation.

4. Press Enter, and the script will execute. The output will be displayed in the command prompt or terminal.

Working with the Python File and Running the Script in Pycharm

1. Running the script: To run the Python script, right-click anywhere within the Python file and select `Run` → `Run ExampleManual.py`. Alternatively, you can use the keyboard shortcut `Shift + F10`. The script will execute, and the output will be displayed in the PyCharm console.

2. Debugging the script: To debug the Python script and set breakpoints for analysis, click on the left gutter of the Python file, next to the line where you want to set the breakpoint. A red dot will appear, indicating the breakpoint. Click on the `Debug` button or use the keyboard shortcut `Shift + F9` to start debugging the script.

3. Interacting with the script: If your script expects user input or provides interactive prompts, you can provide the input in the PyCharm console. The console allows you to interact with the script while it is running.

Modifying the Python File

1. Editing the code: To make changes to the Python code, simply locate the section you want to modify and edit the code accordingly.

2. Saving the changes: PyCharm automatically saves your changes as you work. However, you can manually save the file by going to `File` → `Save` or using the keyboard shortcut `Ctrl + S`.

6.5. Example Code

6.5.1. Introduction

This tutorial walks you through the process of training a neural network model for classifying clothing images, such as sneakers and shirts. It's perfectly fine if you don't grasp every detail immediately; the explanation unfolds dynamically as we progress through this swift overview of a comprehensive TensorFlow program.

The tutorial leverages tf.keras, a high-level API designed for constructing and training models efficiently within the TensorFlow framework.

6.5.2. Environment

The code is implemented in Python using TensorFlow and tf.keras. Numpy and Matplotlib are additionally employed for their utility in numerical operations and data visualization, respectively (See Listing 6.1). See Section 6.4 for installation tutorials.

```
# TensorFlow and tf.keras
import tensorflow as tf

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.1.: Importing the packages

6.5.3. Dataset: Fashion MNIST

The Fashion MNIST dataset comprises 70,000 grayscale images categorized into 10 distinct classes, each depicting individual pieces of clothing in a low-resolution format (28 by 28 pixels). This dataset serves as a seamless substitute for the traditional MNIST dataset, which is often employed as the introductory "Hello, World" for machine learning programs in computer vision. The MNIST dataset features handwritten digits (0, 1, 2, etc.) presented in a format analogous to the clothing items used in this context.

Loading the Dataset

For this task, 60,000 images are utilized to train the neural network, while an additional set of 10,000 images is reserved for evaluating the network's

proficiency in image classification. You can seamlessly access the Fashion MNIST dataset directly through TensorFlow by importing and loading the data from the framework.

The dataset is loaded directly from TensorFlow, split into training and testing sets (See Listing 6.2).

```
fashionMnist = tf.keras.datasets.fashion_mnist
(trainImages, trainLabels), (testImages, testLabels) =
    fashionMnist.load_data()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.2.: Loading the dataset

Class Labels

Upon loading the dataset, four NumPy arrays are obtained:

- `trainImages` and `trainLabels:` These arrays constitute the training set, serving as the data used by the model for learning.
- `testImages` and `testLabels:` These arrays form the test set, employed for assessing the model's performance.

The images in both sets are represented as 28×28 NumPy arrays, where pixel values range from 0 to 255. The corresponding labels are an array of integers ranging from 0 to 9, each signifying the class of clothing that the respective image represents:

Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Assign a singular label to each image. As the dataset lacks class names, save them here for future use in image plotting (See Listing 6.3).

```
classNames = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress',
    'Coat',
                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle
    boot']
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.3.: Assigning a singular label to each image

```
print(trainImages.shape)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.4.: Shape of the training set images

6.5.4. Exploratory Data Analysis

The dataset comprises 60,000 images in the training set, and each image is depicted as a 28 x 28 pixel array (See Listing 6.4):

```
(60000, 28, 28)
```

Each label is represented as an integer ranging from 0 to 9 in Listing 6.5:

```
print(trainLabels)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.5.: Labels of the training set

```
array([9, 0, 0, ..., 3, 0, 5], dtype=uint8)
```

The `dtype=uint8` specifies the data type of the array elements. Specifically, `uint8` stands for "unsigned 8-bit integer."

Moving on to the test set, it consists of 10,000 images, with each image again portrayed as a 28 x 28 pixel array (See Listing 6.6).

```
print(testImages.shape)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.6.: Shape of the test set images

```
(10000, 28, 28)
```

6.5.5. Data Preprocessing

When examining the first image within the training set, you'll observe that the pixel values lie within the range of 0 to 255 (See Listing 6.7 and Figure 6.3).

```
plt.figure()
plt.imshow(trainImages[0])
plt.colorbar()
plt.grid(False)
plt.savefig('preprocessData.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.7.: Pixel values of the first image in the training set.

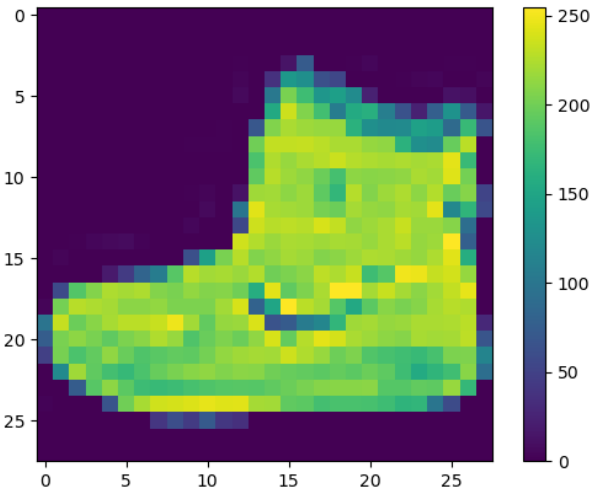


Figure 6.3.: Visualization of the pixel values in the first image of the training set.

Before training the neural network, data preprocessing is performed by scaling pixel values to a range of 0 to 1 (See Listing 6.8). It is crucial to preprocess both the training set and the testing set in a consistent manner.

To ensure that the data is appropriately formatted and that you are prepared to construct and train the network, let's showcase the initial 25 images from the training set. Additionally, we'll display the corresponding class names beneath each image (See Listing 6.9 and Figure 6.4).

```
# Normalize these values to fall within the range of 0 to 1
trainImages = trainImages / 255.0
testImages = testImages / 255.0
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.8.: Data preprocessing to scale pixel values to the range [0, 1].

```
plt.figure(figsize=(10, 10))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(trainImages[i], cmap=plt.cm.binary)
    plt.xlabel(classNames[trainLabels[i]])
plt.savefig('first25TrainingImages.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.9.: Displaying the initial 25 images from the training set with class names.

6.5.6. Building the Model

Creating a neural network involves configuring its layers and subsequently compiling the model. Layers are responsible for extracting representations from the input data, with the expectation that these representations prove insightful for addressing the given problem.

The majority of deep learning involves connecting basic layers in a sequence. Many layers, such as `tf.keras.layers.Dense`, possess parameters that are fine-tuned during the training process (See Listing 6.10).

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.10.: Defining the layers of the neural network model.

The initial layer in this network, `tf.keras.layers.Flatten`, transforms the format of the images from a two-dimensional array (28 by 28 pixels) to a one-dimensional array ($28 \times 28 = 784$ pixels). Visualize

Figure 6.4.: Visualization of the initial 25 images from the training set with corresponding class names.

this layer as the process of unstacking rows of pixels in the image and aligning them in a linear fashion. This layer does not involve learning any parameters; its sole function is to reformat the data.

Following the flattening process, the network comprises a series of two `tf.keras.layers.Dense` layers. These layers are densely connected, or fully connected, neural layers. The initial `Dense` layer consists of 128 nodes (or neurons), while the subsequent (and final) layer produces a logits array with a length of 10. Each node holds a score indicating the likelihood that the current image belongs to one of the 10 classes.

Compiling the Model

Prior to initiating the model training, a few additional configurations are necessary (See Listing 6.11). These adjustments are incorporated during the model's compilation phase:

- **Optimizer:** This determines how the model is updated based on the observed data and its associated loss function.
- **Loss function:** This metric gauges the accuracy of the model

during training, with the objective of minimizing the function to guide the model in the correct direction.

- **Metrics:** These are employed to monitor both the training and testing phases. In the present example, accuracy is utilized as a metric, representing the fraction of images that are accurately classified.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.11.: Compiling the neural network model with optimizer, loss function, and metrics.

6.5.7. Training the Model

Training the neural network model involves the following steps:

1. Feed the training data into the model. In this instance, the training data is stored in the `trainImages` and `trainLabels` arrays.

2. The model learns to establish associations between images and labels.

3. Utilize the trained model to make predictions on a test set—here, represented by the `testcimages` array.

4. Verify the accuracy of predictions by comparing them to the labels in the `testLabels` array.

To train the neural network model see Listing 6.12.

```
model.fit(trainImages, trainLabels, epochs=10)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.12.: Training the neural network model

```
...
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step -
loss: 0.2481 - accuracy: 0.9072
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step -
loss: 0.2392 - accuracy: 0.9102
```

Throughout the training process, the metrics of loss and accuracy are presented. This particular model attains an accuracy of approximately 91% on the training data. Note that without the normalizing step in Listing 6.8, the accuracy was about 80%. This shows the effect of this step in the process.

A loss of 0.2392 represents the average loss on the training dataset for the current epoch. The loss is a measure of how well the model is performing, with lower values indicating better performance.

Evaluation

Evaluate the model's performance on the test dataset in Listing 6.13:

```
testLoss, testAcc = model.evaluate(testImages, testLabels,
    verbose=2)

print('\nTest accuracy:', testAcc)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.13.: Evaluating the neural network model on the test dataset.

```
313/313 - 1s - loss: 0.3383 - accuracy: 0.8817 - 556ms/epoch -
2ms/step

Test accuracy: 0.8816999793052673
```

It turns out that the accuracy on the test dataset is a little less than the accuracy on the training dataset. This gap between training accuracy and test accuracy represents overfitting. Overfitting happens when a machine learning model performs worse on new, previously unseen inputs than it does on the training data. An overfitted model "memorizes" the noise and details in the training dataset to a point where it negatively impacts the performance of the model on the new data.

6.5.8. Making Predictions

Having trained the model, you can employ it to generate predictions for certain images (See Listing 6.14). Append a softmax layer to transform the model's linear outputs—logits—into probabilities, facilitating a more intuitive interpretation.

```
probabilityModel = tf.keras.Sequential([model, tf.keras.
    layers.Softmax()])
predictions = probabilityModel.predict(testImages)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.14.: Making predictions using the trained neural network model

```
313/313 [==============================] - 0s 1ms/step
```

The model has provided predictions for the label of each image in the testing set. Let's examine the first prediction in Listing 6.15:

```
print(predictions[0])
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.15.: producing the prediction array for the first image in the testing set

```
[1.6619989e-08 1.4881327e-09 8.5905718e-09 4.6579121e-08 7.5430655e-08
6.0041228e-05 2.0011402e-07 2.5325418e-03 6.7323890e-07 9.9740642e-01]
```

A prediction comprises an array of 10 numbers, indicating the model's "confidence" in associating the image with each of the 10 distinct articles of clothing. Identify the label with the highest confidence value:

Hence, the model expresses the highest confidence in categorizing this image as an ankle boot, corresponding to the `classNames[9]`. Verifying the test label confirms the accuracy of this classification (See Listing 6.16):

```
print(np.argmax(predictions[0]))
print(testLabels[0])
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.16.: Verifying the predicted and actual labels for the image.

```
9
9
```

6.5.9. Verify Predictions

After training the model, you can utilize it to make predictions on specific images (See Listings 6.17 and 6.18 and Figures 6.5 and 6.6). Let's examine the predictions for the 0th and 12th images, showcasing correct predictions in blue and incorrect predictions in red.

```
i = 0
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plotImage(i, predictions[i], testLabels, testImages)
plt.subplot(1, 2, 2)
plotValueArray(i, predictions[i], testLabels)
plt.savefig(f'predictionPlot{i}.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.17.: Code for visualizing predictions on the 0th image

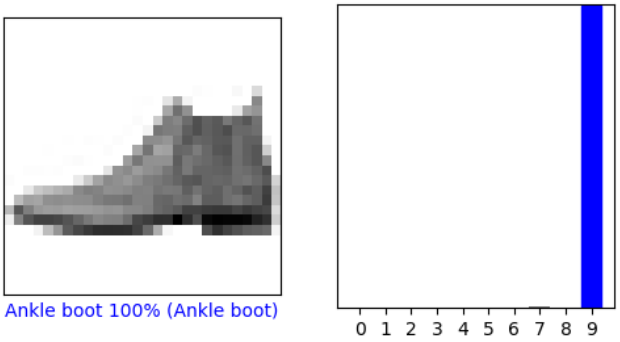


Figure 6.5.: Predictions for the 0th image. Correct prediction is in blue.

```
i = 12
plt.figure(figsize=(6, 3))
plt.subplot(1, 2, 1)
plotImage(i, predictions[i], testLabels, testImages)
plt.subplot(1, 2, 2)
plotValueArray(i, predictions[i], testLabels)
plt.savefig(f'predictionPlot{i}.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.18.: Code for visualizing predictions on the 12th image

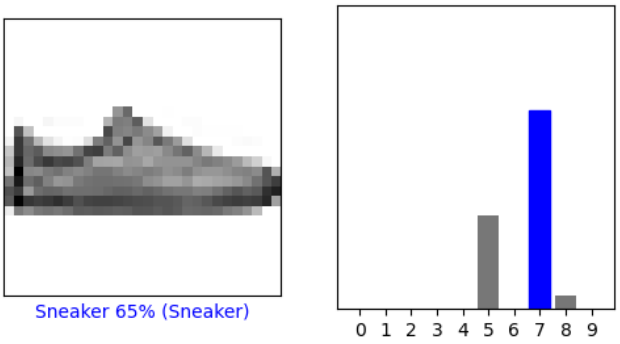


Figure 6.6.: Predictions for the 12th image. Correct prediction is in blue

Visualize multiple images along with their corresponding predictions is shown in Listing 6.19. The result is shown in 6.7.It's important to acknowledge that the model may make errors despite exhibiting high confidence.

```
numRows = 5
numCols = 3
numImages = numRows * numCols
plt.figure(figsize=(2 * 2 * numCols, 2 * numRows))
for i in range(numImages):
    plt.subplot(numRows, 2 * numCols, 2 * i + 1)
    plotImage(i, predictions[i], testLabels, testImages)
    plt.subplot(numRows, 2 * numCols, 2 * i + 2)
    plotValueArray(i, predictions[i], testLabels)
plt.tight_layout()
plt.savefig('predictionPlots.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.19.: Code for visualizing predictions for multiple images

6.5.10. Use the Trained Model

Finally, leverage the trained model to predict a single image (See Listing 6.20).

```
(28, 28)
```

Optimized for batch processing, tf.keras models excel at making predictions on collections of examples simultaneously. Therefore, even when working with a single image, it is necessary to include it in a list (See Listing 6.21).

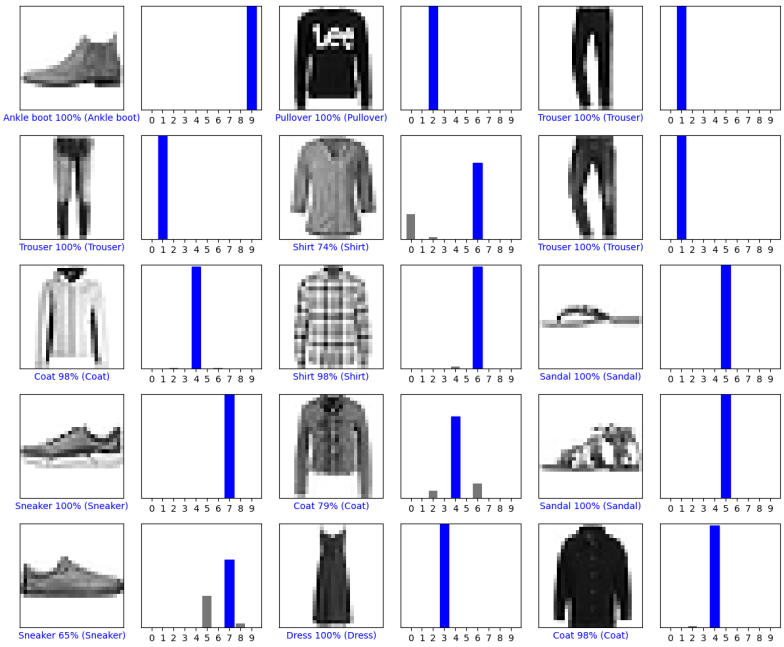


Figure 6.7.: Visualizing predictions for multiple images

```
img = testImages[1]

print(img.shape)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.20.: Code for predicting a single image

```
img = (np.expand_dims(img, 0))
print(img.shape)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.21.: Preparing a single image for prediction by adding it to a list.

```
(1, 28, 28)
```

Predicting the accurate label for this image is shown in 6.22.

```
1/1 [==============================] - 0s 23ms/step
[[8.4762414e-06 2.3953922e-11 9.9964368e-01 1.5056546e-11 3.0798238e-04
8.6492816e-16 3.9860388e-05 3.5954907e-23 6.0896788e-11 4.6433261e-12]]
```

To display the predicted probabilities for each class see Listing 6.23 and Figure 6.8.

```
predictionsSingle = probabilityModel.predict(img)
print(predictionsSingle)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.22.: Making predictions for a single image.

```
plotValueArray(1, predictionsSingle[0], testLabels)
_ = plt.xticks(range(10), classNames, rotation=45)
plt.savefig('singlePredictionPlot.png')
plt.show()
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.23.: Displaying the predicted probabilities for each class.

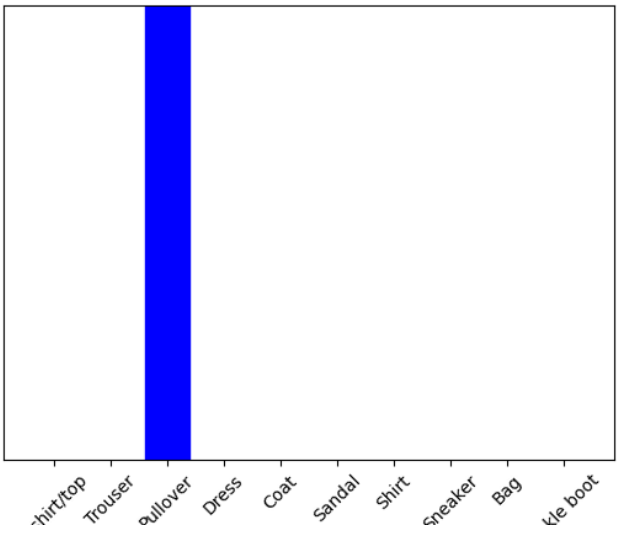


Figure 6.8.: Prediction for a single image

```
print(np.argmax(predictionsSingle[0]))
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.24.: Extracting predictions for the sole image in the batch

See Listing 6.24 to extract the predictions for the sole image in the batch:

```
(1, 28, 28)
```

6.5.11. Saving the Model

A TensorFlow model is saved in the SavedModel format using the `model.save()` method, with the saved path specified as `"./savedModel"` (See Listing 6.25). Following this, the SavedModel is converted to the TensorFlow Lite format using `tf.lite.TFLiteConverter.from_saved_model()`. The resulting TensorFlow Lite model is then saved to a file, and the path for the TensorFlow Lite model is set as `"./savedModel/model.tflite"`. This process is essential for deploying the model in resource-constrained environments, such as mobile or edge devices, where the lightweight TensorFlow Lite format enhances efficient model inference.

```
# Save the model in the SavedModel format
savedModelPath = "./savedModel"
model.save(savedModelPath)

# Convert the SavedModel to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_saved_model(
    savedModelPath)
tfliteModel = converter.convert()

# Save the TFLite model to a file
tfliteModelPath = "./savedModel/model.tflite"
with open(tfliteModelPath, "wb") as f:
    f.write(tfliteModel)
```

Code/TensorFlow/clothingimgClassification.py

Listing 6.25.: Saving the TensorFlow model in the SavedModel format and converting it to TensorFlow Lite

6.6. Further Readings

- The article "Deep Learning With TensorFlow: A Review" [PNW20] explores the core concepts of TensorFlow, a leading deep learning library initially developed by Google. It highlights TensorFlow's widespread popularity and applications, particularly in educational and behavioral sciences. The review covers key neural network models and optimization methods, emphasizing TensorFlow's role in simplifying implementation challenges. Essential TensorFlow features, such as graph construction functions and TensorBoard visualization, are discussed, with a practical application example of training a convolutional neural network for handwritten digit classification. The article concludes by comparing low- and high--level APIs, touching on GPU support, distributed training, and the TensorFlow Probability library for probabilistic modeling.

- "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" [Gér22] is an accessible guide for beginners in machine learning. The book aims to provide readers with the concepts and tools needed to implement programs that learn from data. Scikit-Learn is introduced for its simplicity and efficiency, while TensorFlow is explored for distributed numerical computation, particularly in training large neural networks. The high-level deep learning API, Keras, integrated with TensorFlow, simplifies the training and execution of neural networks. With a hands-on approach, the book emphasizes practical examples and minimal theory to foster an intuitive understanding of machine learning concepts.

- The article "A Tour of TensorFlow" [Gol16] provides a thorough review of TensorFlow. The paper contextualizes TensorFlow within the realm of modern deep learning concepts and software, covering its computational paradigms, distributed execution model, programming interface, and visualization tools. Goldsborough conducts a qualitative and quantitative comparison with alternative libraries, discussing the rise of deep learning in recent years. The review concludes by exploring real-world use-cases of TensorFlow in academia and industry, offering a comprehensive overview of its contributions to machine learning.

- The book "Introduction to TensorFlow 2.0" [SM19] is a practical guide that delves into the key changes made by Google's TensorFlow team. The book covers data processing, building machine learning models, including neuro-linguistic programming, and deploying models in production. Aimed at data analysts, engineers, and newcomers to data science and machine learning, the book stands out for its simplicity, real-world applications, and inclusion of case studies. Overall, it provides valuable insights into TensorFlow 2.0's fundamentals and practical use.

7. NumPy package

7.1. Introduction

NumPy is a freely available Python library that finds extensive application across various scientific and engineering disciplines. It has become the de facto standard for handling numerical data in Python and forms the backbone of the scientific Python and PyData ecosystems. NumPy caters to a wide user base, from novices in coding to seasoned researchers engaged in cutting-edge scientific and industrial Research and Development. The NumPy API is heavily utilized in numerous data science and scientific Python packages, including but not limited to Pandas, SciPy, Matplotlib, scikit-learn, and scikit-image.[Idr15]

The NumPy library is equipped with data structures for multidimensional arrays and matrices. It offers ndarray, a homogeneous n-dimensional array object, along with methods for efficient operations. NumPy enables a broad range of mathematical operations on arrays. It enhances Python with robust data structures, ensuring efficient computations with arrays and matrices. Furthermore, it provides a vast library of high-level mathematical functions that work on these arrays and matrices.

7.2. Description

NumPy is the cornerstone of scientific computing in Python. It's a Python library that offers a multidimensional array object, along with various related objects like masked arrays and matrices. It also provides a plethora of routines for rapid operations on arrays, encompassing mathematical, logical, shape manipulation, sorting, selection, I/O, discrete Fourier transforms, elementary linear algebra, basic statistical operations, random simulations, and much more.[Idr15]

The heart of the NumPy package is the `ndarray` object. This object encapsulates n-dimensional arrays of uniform data types, with a multitude of operations executed in compiled code for efficiency. There are several key distinctions between NumPy arrays and conventional Python sequences:

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are

executed more efficiently and with less code than is possible using Python's built-in sequences.

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an `ndarray` will create a new array and delete the original.

- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

An increasing number of scientific and mathematical packages based on Python are utilizing NumPy arrays. While these packages generally accept input in the form of Python sequences, they convert such input into NumPy arrays for processing and often return results as NumPy arrays. This implies that to effectively use a significant portion of today's scientific/mathematical software based on Python, mere knowledge of Python's built-in sequence types is not enough - proficiency in using NumPy arrays is also required.

7.2.1. Features of Numpy

- Open-source and community-driven: NumPy is an open-source project with a large and active community of developers and users, continuously maintained and improved with regular releases.

- Array indexing and slicing: NumPy provides advanced indexing and slicing capabilities for accessing and manipulating elements within arrays, facilitating data manipulation.

- Random number generation: NumPy provides tools for generating random numbers from various distributions, as well as functions for shuffling arrays and generating random samples.

- Broadcasting: NumPy supports broadcasting, allowing arrays with different shapes to be combined in arithmetic operations, making code concise and intuitive.

- Element-wise operations: NumPy offers a comprehensive set of mathematical functions for element-wise operations on arrays, including arithmetic, trigonometric, and logarithmic functions.

- Vectorized operations: NumPy enables efficient computation by performing operations on entire arrays at once, eliminating the need for explicit looping over individual elements.

- Linear algebra operations: NumPy includes functions for essential linear algebra operations such as matrix multiplication, decomposition, eigenvalue computation, and solving linear equations.
- Integration with other libraries: NumPy seamlessly integrates with other scientific computing libraries in Python, such as SciPy, matplotlib, and pandas, facilitating complex computational workflows.
- Efficient memory management: NumPy's array data structure is implemented in C, allowing for efficient memory management and low-level optimizations, making it suitable for handling large datasets.
- N-dimensional array object (`ndarray`): NumPy provides a powerful array object, `ndarray`, which allows efficient storage and manipulation of large datasets in multiple dimensions.

7.3. Installation

Python is the sole requirement for setting up NumPy. If you're new to Python and seeking the easiest route, we suggest the Anaconda Distribution. It not only comes with Python and NumPy, but also includes a variety of other packages frequently used in scientific computing and data science.

NumPy can be installed with `conda`, with `pip`, with a package manager on macOS and Linux, or from source. For more detailed instructions, consult our Python and NumPy installation guide below.

Handling packages can be a difficult task, hence the existence of numerous tools. A variety of tools are available that complement pip for web and general Python development. Spack is a notable choice for high-performance computing (HPC). However, for the majority of NumPy users, conda and pip are the preferred tools.

Pip & conda

Pip and conda are the primary tools for installing Python packages. They have some overlapping functionalities (for instance, both can install numpy), but they can also function in tandem. Understanding the key differences between pip and conda is crucial for effective package management.

The initial distinction is that conda is a cross-language tool capable of installing Python, whereas pip is specific to a particular Python installation on your system and only installs packages to that Python installation. This implies that conda can install non-Python libraries and tools that might be necessary like compilers, a capability that pip lacks.

Another distinction is that pip sources installations from the Python Packaging Index (PyPI), whereas conda uses its own channels, usually 'defaults' or 'conda-forge'. Although PyPI boasts the most extensive package collection, all commonly used packages can also be found in conda.

A third distinction is that conda provides a comprehensive solution for handling packages, dependencies, and environments. In contrast, when using pip, you might require an additional tool to manage environments or intricate dependencies.

If you use pip, you can install NumPy with:

```
pip install numpy
```

Code/Numpy/NumPy.py

Listing 7.1.: Installing NumPy using PIP

If you use conda, you can install NumPy from the defaults or conda-forge channels:

```
conda create -n my-env
conda activate my-env
# If you want to install from conda-forge
conda config --env --add channels conda-forge
# The actual install command
conda install numpy
```

Code/Numpy/NumPy.py

Listing 7.2.: Installing NumPy using terminal

7.4. Example - Manual

Installation and Setup

1. **Download and install Python:**

   Visit the official Python website `https://www.python.org` and download the latest version of Python for your operating system. Follow the installation instructions provided.

2. **Install PyCharm:**

Visit the JetBrains website `https://www.jetbrains.com/pycharm` and download PyCharm Community Edition, which is the free version. Install PyCharm by following the installation instructions specific to your operating system.

Opening the Python File in PyCharm

1. Launch PyCharm: Open the PyCharm application from your desktop or applications menu.

2. Create a new project: Click on `Create New Project` or go to `File` → `New Project`. Choose a suitable location for your project and provide a name.

3. Open the Python file: Once the project is created, navigate to the project directory in the PyCharm project view. Right-click on the desired folder and select `New` → `Python File`. Provide a name for the file and click `OK`.

4. Copy your Python code: Open your Python file (with .py extension) in a text editor and copy the contents.

5. Paste the code: Paste the copied code into the newly created Python file in PyCharm.

Working with the Python File

1. Running the script: To run the Python script, right-click anywhere within the Python file and select `Run` → `Run ExampleManual.py`. Alternatively, you can use the keyboard shortcut `Shift + F10`. The script will execute, and the output will be displayed in the PyCharm console.

2. Debugging the script: To debug the Python script and set breakpoints for analysis, click on the left gutter of the Python file, next to the line where you want to set the breakpoint. A red dot will appear, indicating the breakpoint. Click on the `Debug` button or use the keyboard shortcut `Shift + F9` to start debugging the script.

3. Interacting with the script: If your script expects user input or provides interactive prompts, you can provide the input in the PyCharm console. The console allows you to interact with the script while it is running.

Modifying the Python File

1. Editing the code: To make changes to the Python code, simply locate the section you want to modify and edit the code accordingly.
2. Saving the changes: PyCharm automatically saves your changes as you work. However, you can manually save the file by going to `File` → `Save` or using the keyboard shortcut `Ctrl + S`.

Further Assistance and Resources

- PyCharm Documentation: PyCharm offers comprehensive documentation to help you understand its features and functionality. You can access it online at `https://www.jetbrains.com/help/pycharm`.
- Python Documentation: The official Python documentation provides detailed information about the Python language, libraries, and best practices. It is available at `https://docs.python.org`.
- Online Python Communities: Joining online communities like Stack Overflow `https://stackoverflow.com` or the Python subreddit `https://www.reddit.com/r/Python` can provide valuable insights and assistance from experienced Python developers.

7.4.1. Importing NumPy

The code begins by importing the required libraries, numpy (`np`).

```
import numpy as np
```

Code/Numpy/NumPy.py

Listing 7.3.: Importing the package

7.4.2. Verison Check

You can check the version of Numpy by using the following command:

```
import numpy
print(numpy.__version__)
```

Code/Numpy/NumPy.py

Listing 7.4.: Version Check

You can save and load numpy array into a file with numpy functions such as follows: “numpy.save” and “numpy.load”.

```
import numpy as np
arr = np.array([1, 2, 3, 4]) np.save('my_array', arr)
loaded_array = np.load('my_array.npy')
```

Code/Numpy/NumPy.py

Listing 7.5.: How to load and save NumPy Files

7.4.3. Arrays

- Creating arrays is the foundation of NumPy. You can create arrays using various methods like `np.array()`, `np.zeros()`, `np.ones()`, `np.arange()`, and `np.linspace()`.
- NumPy allows you to perform various mathematical operations on arrays, including arithmetic operations, trigonometric functions, exponential functions, and more.
- NumPy provides powerful indexing and slicing capabilities to access and manipulate elements within arrays.
- NumPy provides functions to change the shape and dimensions of arrays.

```
#Creating Array

arr_1d = np.array([1, 2, 3, 4, 5])
arr_zeros = np.zeros((2, 3))
arr_identity = np.eye(3)
arr_range = np.arange(1, 10, 2)
arr_linspace = np.linspace(0, 1, 5)

#Array Operations
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
result_add = arr1 + arr2
result_mul = arr1 * arr2
result_dot = np.dot(arr1, arr2)
sin_arr = np.sin(arr1)
```

Code/Numpy/NumPy.py

Listing 7.6.: Arrays creation and operations

7.4.4. Importing and exporting data

In NumPy, you can import and export data from various file formats such as text files, CSV files, and binary files.Depending on the data format and

```
#Indexing and Slicing in Arrays
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
element = arr[1, 2]
row_slice = arr[1]
col_slice = arr[:, 1]
bool_index = arr[arr > 5]
```

Code/Numpy/NumPy.py

Listing 7.7.: Indexing and Slicing in Arrays

```
#Shape Manipulation in Arrays
arr = np.array([[1, 2, 3], [4, 5, 6]])
reshaped_arr = arr.reshape(3, 2)
transposed_arr = arr.T
flattened_arr = arr.flatten()
```

Code/Numpy/NumPy.py

Listing 7.8.: Shape Manipulation in Arrays

requirements, you may need to customize the import/export functions accordingly. Here are examples of importing and exporting data using NumPy:

```
# Load data from a CSV file
data = np.genfromtxt('data.csv', delimiter=',')
# Load data from a NumPy binary file
data = np.load('data.npy')

# Exporting Data

# Exporting data to a text file
# Assuming 'data' is your NumPy array
np.savetxt('data.txt', data, delimiter=',')
# Exporting data to a CSV file
np.savetxt('data.csv', data, delimiter=',')
# Exporting data to a NumPy Binary file
np.save('data.npy', data)
```

Code/Numpy/NumPy.py

Listing 7.9.: Importing and exporting data in NumPy

7.4.5. Linear Algebra Operations:

NumPy provides an extensive array of functions for performing linear algebra operations.

```
#Linear Algebra Operations

arr = np.array([[1, 2], [3, 4]])
determinant = np.linalg.det(arr)
inverse = np.linalg.inv(arr)
eigenvalues, eigenvectors = np.linalg.eig(arr)

# Solving linear equations

A = np.array([[2, 3], [1, -1]])
b = np.array([8, 1])
solution = np.linalg.solve(A, b)
```

Code/Numpy/NumPy.py

Listing 7.10.: Linear Algebra Operations

7.4.6. Error Handling in NumPy

Like many Python libraries, NumPy incorporates error handling mechanisms to manage exceptions and errors that might occur during numerical computations and array operations. These methods are vital for identifying and resolving problems that occur during data manipulation and analysis. Here are some common error handling techniques in NumPy:

- Error Handling Functions: Functions like `np.seterr()` in NumPy can be used to manage error handling modes, such as raising exceptions, issuing warnings, or completely ignoring errors.
- Error Handling with Arrays: Functions for checking array bounds and detecting special values like `np.isnan()` and `np.isinf()` are used to avoid errors and manage exceptional conditions when working with NumPy arrays.
- Try-Except Blocks: These blocks are used to catch and manage specific exceptions during NumPy operations, ensuring the code runs smoothly even when errors occur.
- Error Reporting: NumPy offers detailed error messages and tracebacks to help identify the location and nature of errors, which aids in effective debugging and problem-solving.

7.5. Further Reading

- **NumPy Official Documentation:** This is an invaluable resource for understanding NumPy's functions, methods, and usage patterns.

It provides detailed explanations and examples for each aspect of the library. `https://numpy.org/doc/stable/`

- **NumPy User Guide:** This is the official guide from NumPy, providing extensive documentation on its features, including array creation, manipulation, and mathematical operations. `https://numpy.org/doc/stable/user/index.html`

Part IV.

KDD Process

8. Knowledge Discovery in Databases (KDD) Process

8.1. Introduction

KDD plays a pivotal role in helping organizations navigate the overwhelming volumes of data generated in today's information-driven world [FPSS96]. This process involves transforming raw data into actionable knowledge, uncovering hidden patterns, and providing valuable insights for decision-making. As we delve into the intricacies of the KDD framework, it becomes evident that this multidisciplinary approach has transformative potential across diverse sectors.

This chapter aims to provide an understanding of the KDD process, shedding light on its challenges, and the critical role it plays in harnessing the power of data to inform strategic and informed decision-making.

8.2. Impracticality of Manual Data Analysis

Traditional manual methods of data analysis and interpretation are becoming impractical as data volumes grow exponentially. With databases containing billions of records and numerous fields, human capacities are surpassed. Health-care specialists analyzing trends and planetary geologists cataloging geologic objects are few of the examples [FPSS96]. The need for automation in knowledge extraction from large databases becomes evident.

8.3. The KDD Process Framework

The KDD process, as depicted in the Figure 8.1, begins with a comprehensive understanding of the problem domain. This initiates two main phases: "Deployment" and "Development."

In the "Deployment" phase, databases serve as the repository for relevant data. The arrow from "Databases" to "Data Selection" signifies the crucial step of selecting relevant data for analysis. at the same time, the "Monitoring" step oversees the ongoing processes, including the monitoring of new data, model inference, and results. This cyclic monitoring ensures adaptability to changing data conditions.

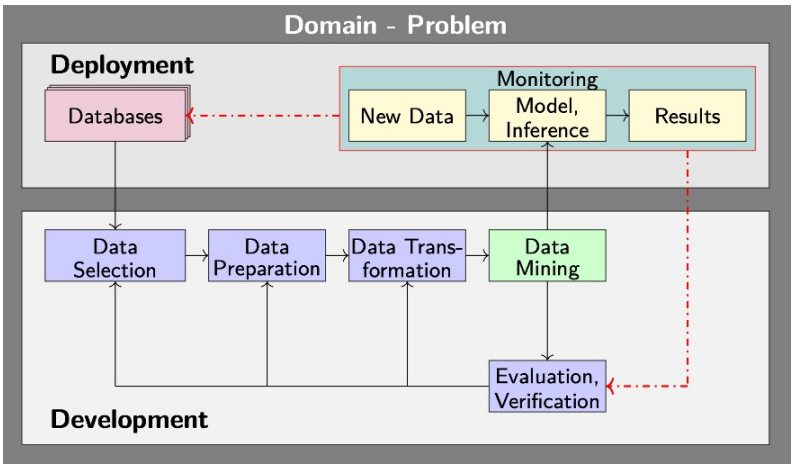


Figure 8.1.: KDD Process Workflow [Win23].

In the "Development" phase, a sequence of activities unfolds, starting with "Data Selection," followed by "Data Preparation," "Data Transformation," and ultimately "Data Mining." The arrows connecting these stages denote the sequential progression of operations.

From the "Data Mining" stage, two arrows extend: one leading to "Model Inference" and another to "Evaluation (Verification)." "Model Inference" involves deriving models from the mined data, while "Evaluation" assesses the performance and validity of these models.

The cyclical nature of the KDD process is highlighted through three feedback arrows from "Evaluation (Verification)" back to the initial stages of "Data Selection," "Data Preparation," and "Data Transformation." This iterative loop emphasizes continuous refinement based on the evaluation outcomes, fostering an adaptive and improving analytical process.

The "Monitoring" step contributes to the iterative nature of the process by providing feedback to "Databases," ensuring that the ongoing monitoring influences the selection of data from databases. Additionally, the "Results" step influences "Evaluation (Verification)," reinforcing the importance of the outcomes in guiding further verification or refinement.

8.4. Challenges in Data Mining

Data mining faces challenges such as impracticality due to massive datasets and high dimensionality, user interaction, overfitting, missing data, and the need for understandable patterns [FPSS96]. Integrating domain knowledge and addressing issues related to changing data, data types, and multimedia content are of high importance.

8.5. Why KDD Process for Speech Recognition?

The Knowledge Discovery in Databases (KDD) process plays a crucial role in keyword spotting for several reasons:

1. **Data Abundance and Complexity:** The Arduino Nano 33 BLE Sense is likely to generate substantial amounts of sensor data. KDD provides a systematic approach to handling large volumes of diverse data, allowing efficient extraction of meaningful patterns and insights.

2. **Feature Extraction and Selection:** Keyword spotting involves identifying specific patterns within the collected data. KDD assists in selecting and extracting relevant features from raw sensor data, optimizing the data for accurate keyword detection.

3. **Adaptability to Changing Conditions:** The iterative nature of KDD, with feedback loops and continuous monitoring, is valuable for adapting the model to changing conditions. This ensures sustained performance over time.

4. **Evaluation and Verification:** KDD emphasizes evaluating models to ensure their effectiveness. In keyword spotting, it ensures the model performs accurately, providing insights for improvement and fine-tuning.

5. **Interpretability and Understandability:** KDD places emphasis on finding patterns that are not only accurate but also understandable. This is crucial in keyword spotting for developing transparent and interpretable models.

8.6. Conclusion

Despite its rapid growth, the field of KDD is still in its infancy [FPSS96]. A balanced approach is needed to be taken, ensuring that the potential contributions of KDD are not overstated, and users understand both the capabilities and limitations of KDD tools. While challenges persist, the societal and economic needs for managing and analyzing vast amounts of data continue to drive the growth of KDD, making it a field with significant potential for future developments.

9. Data Description

9.1. Introduction

The Speech Commands dataset aims to establish a standardized training and assessment dataset designed for straightforward speech recognition tasks. Its principal objective is to facilitate the development and testing of compact models capable of identifying instances when a specific word is uttered from a predefined set of ten words or fewer. The goal is to minimize false positives originating from background noise or irrelevant speech, a common challenge referred to as keyword spotting [War18].

9.2. License

In order to broaden its accessibility among researchers and developers, this dataset has been made available under the Creative Commons BY 4.0 license[5]. This allows for seamless integration into tutorials and scripts, enabling users to download and utilize it without the need for any user intervention, such as website registration or obtaining permission from an administrator via email. Additionally, the familiarity of this license in commercial contexts often expedites the approval process by legal teams when necessary [War18].

9.3. Related Wok

Mozilla's Common Voice dataset boasts over 500 hours of recordings from 20,000 diverse contributors and is released under the Creative Commons Zero license, akin to the public domain. This licensing structure facilitates easy integration and expansion. The dataset is sentence-aligned and was generated through volunteers reading requested phrases via a web application [War18].

LibriSpeech [Pan+15] comprises 1,000 hours of English speech, made available under the Creative Commons BY 4.0 license, and utilizes the widely supported open-source FLAC encoder for storage. Though labeled at the sentence level only, it lacks word-level alignment, making it more suitable for full automatic speech recognition rather than keyword spotting [War18].

TIDIGITS encompasses 25,000 digit sequences spoken by 300 contributors in a quiet environment. This dataset is exclusively accessible under a commercial license from the Language Data Consortium and is stored in the NIST SPHERE file format, posing challenges in decoding with modern software. Initial experiments on keyword spotting were conducted using this dataset [War18].

9.4. Dataset Characteristics, Collection and Origin

The distinctions between on-device keyword spotting and general speech recognition models lead to significant differences in their training and evaluation processes. While datasets like Mozilla's Common Voice show promise for general speech tasks, their adaptability to keyword spotting is challenging [War18]. Some characteristics of the data are:

- Realistic Audio Capture
  - Avoiding the use of studio-captured samples. Reasons:
    - Absence of background noise in the audio
    - Recorded with high-quality microphones
- Recorded using phone or laptop microphones
- Language focus: English
- Audio format: WAV
- Various accents in addition to American English
- Several individuals are recorded (to ensure speaker independence)
- One second single words
- Sample rate (frequency) set to 16 KHz
- Download size: 2.37 GB
- Dataset size: 8.17 GB
- Subset of the dataset used: 415 MB
- Open-source
- Anonymous
  - Recording personally identifiable information was avoided

Using studio-captured samples appeared impractical due to the absence of background noise, high-quality microphones, and a formal setting. Successful models needed to handle noisy environments, low-quality recording equipment, and natural, casual conversation. To address this, all utterances were recorded using phone or laptop microphones in various user locations. The only exception was when users were requested to be alone in a closed room to avoid background conversations for privacy reasons [War18].

Aiming to counter the common bias towards American English prevalent in many voice interfaces, a wide variety of accents was sought to be gathered [War18].

Another objective was to have as many different people recorded as possible. Speaker independence significantly enhances the utility of keyword-spotting models [War18].

The avoidance of recording any personally identifiable information from contributors was also desired, given that extreme care is required for handling such data due to privacy reasons. This involved refraining from requesting attributes like gender or ethnicity, not mandating a sign-in through a user ID that could link to personal data, and necessitating users to agree to a data-usage agreement before contributing [War18].

To streamline the training and evaluation process, the decision was made to confine all utterances to a standardized duration of one second. This involves excluding longer words, but given that the typical targets for keyword recognition are short, this restriction did not appear excessively limiting. Furthermore, it was determined to exclusively capture single words spoken in isolation, as opposed to within sentences. This choice closely aligns with the targeted trigger word task and simplifies the labeling process, as alignment becomes less crucial [War18].

To download the dataset: `https://storage.googleapis.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz`

9.5. Selection of Words

Decision was made to maintain a limited vocabulary for the purpose of ensuring a lightweight capture process, while still incorporating sufficient variety for potential usefulness in certain applications by models trained on the data. This led to the selection of twenty common words as the core vocabulary, encompassing digits zero to nine and ten additional words suitable for IoT or robotics commands: "Yes," "No," "Up," "Down," "Left," "Right," "On," "Off," "Stop," "Go," "Backward," "Forward," "Follow," and "Learn." Addressing a key challenge in keyword recognition, a set of words was needed to act as tests for the ability to ignore speech lacking triggers. Some, like "Tree," were chosen for their phonetic similarity to target words,

serving as tests for a model's discernment, while others were arbitrarily selected as short words covering various phonemes, resulting in the final list: "Bed," "Bird," "Cat," "Dog," "Happy," "House," "Marvin," "Sheila," "Tree," and "Wow." The final list of all the utterances is shown in Table 9.1.

9.6. Quality Control

Criteria were needed to accept or reject submissions due to the variable quality of the gathered audio utterances. An informal guideline was applied, where clips that were indiscernible to a human listener or sounded like incorrect words were slated for [War18].

To eliminate extremely short or quiet clips, the nature of the OGG compression format was leveraged. Clips with minimal audio content had significantly smaller file sizes, and a heuristic was established to reject files smaller than 5 KB [War18].

Subsequently, the OGG files were transformed into uncompressed WAV files containing PCM sample data at 16KHz, as this format facilitates further processing [War18].

Samples from other sources, which arrived as WAV files with varying sample rates, were also resampled to 16 KHz WAV files [War18].

9.7. Capture the Loudest Segment

The existence of a considerable number of utterances that were too quiet or entirely silent was noted during manual inspection of the samples gathered. The alignment of spoken words within the 1.5-second file was found to be arbitrary, depending on the user's response speed to the displayed word. To address both issues, a simple audio processing tool was created to assess the overall volume of the clips [War18].

In the initial stage, the absolute differences of all the samples from zero were summed, using a scale where -32768 in the 16-bit sample data represented -1.0 as a floating-point number, and +32767 was 1.0. The mean average of this value was examined to estimate the overall volume of the utterance. Through experimentation, it was determined that anything below 0.004 on this metric was likely too quiet to be intelligible, leading to the removal of such clips [War18].

To approximate the correct alignment, the tool then extracted the one-second clip containing the highest overall volume. This approach tended to center the spoken word in the middle of the trimmed clip, assuming that the utterance was the loudest part of the recording [War18].

9.8. Release Procedure

The recorded utterances were organized into individual folders, each corresponding to a specific word. The original 16-digit hexadecimal speaker ID numbers present in the web application's file names were hashed into 8-digit hexadecimal IDs. Speaker IDs from other sources, such as paid crowdsourcing sites, were similarly hashed into the same format. This was done to eliminate any link to worker IDs or other personally identifiable information. While the hash function used is stable, ensuring consistency, the IDs for existing files should remain unchanged in future releases even with the addition of more speakers [War18].

9.9. Background Noise

A crucial requirement for keyword spotting in real products is the ability to differentiate between audio containing speech and clips devoid of speech. To aid in training and testing this capability, several minute-long 16 KHz WAV files of various types of background noise were incorporated. Some of these were directly recorded from noisy environments, such as those near running water or machinery, while others were mathematically generated using Python. In order to differentiate these files from word utterances, they were placed within a designated `"_background_noise_"` folder located at the root of the archive [War18].

9.10. Potential Anomalies in the Dataset

In the preceding sections, we discussed the handling of anomalies. This section provides a summary of potential anomalies that could be introduced in such datasets. Some common anomalies or challenges might include:

Background Noise Variability

- **Anomaly:** Inconsistencies in background noise levels or types across different recordings.
- **Challenge:** Ensuring the model can effectively distinguish between target keywords and various background noises.

Accents and Pronunciation

- **Anomaly:** Presence of diverse accents or variations in pronunciation.

- **Challenge:** Ensuring the model's robustness to different accents and pronunciations, preventing bias towards specific linguistic patterns.

Recording Quality

- **Anomaly:** Variability in recording quality, including low-quality or noisy recordings.

- **Challenge:** Developing a model that performs well on recordings from different devices and environments.

Speaker Characteristics

- **Anomaly:** Differences in speaker characteristics such as pitch, speed, or age.

- **Challenge:** Designing a model that generalizes well across various speaker attributes.

Imbalanced Dataset

- **Anomaly:** Significant variations in the number of instances for different keywords.

- **Challenge:** Ensuring a balanced dataset to prevent the model from being biased toward more frequently occurring keywords.

Unintended Triggers

- **Anomaly:** Instances where unintended words or sounds trigger the keyword spotting.

- **Challenge:** Implementing measures to minimize false positives caused by words with phonetic similarities or ambiguous sounds.

Inconsistent Recording Conditions

- **Anomaly:** Changes in recording conditions, such as recording in a noisy environment for some words and a quiet room for others.

- **Challenge:** Ensuring the model's ability to handle diverse recording conditions.

Data Privacy Concerns

- **Anomaly:** Accidental inclusion of personally identifiable information in recordings.
- **Challenge:** Implementing strict protocols to ensure privacy and anonymizing data appropriately.

Unexpected Word Variations

- **Anomaly:** Variability in word variations (e.g., different verb tenses, plural forms).
- **Challenge:** Creating a model that recognizes variations of target words.

Mislabeling or Incorrect Annotations

- **Anomaly:** Instances where labels or annotations do not accurately represent the content.
- **Challenge:** Implementing rigorous quality control to identify and rectify mislabeled instances.

9.11. How to Record your Own voice

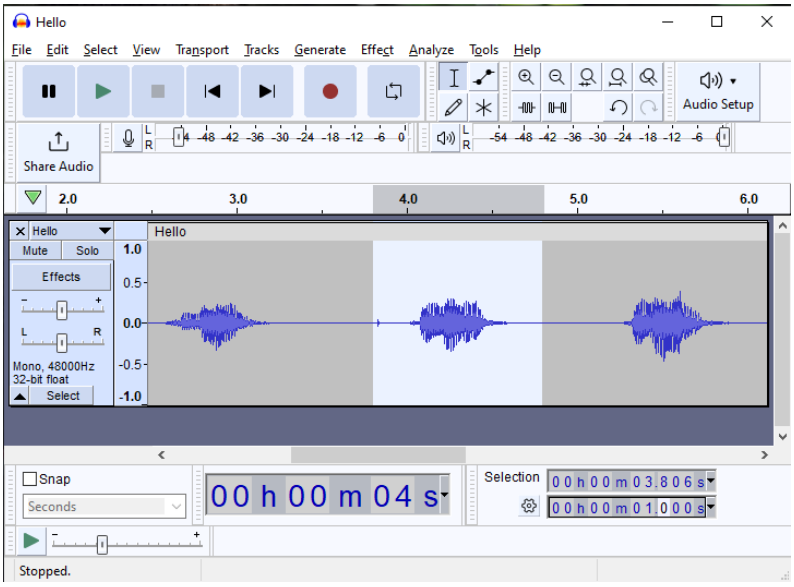


Figure 9.1.: Audacity user interface

1. Record your voice using a recording device.
2. Change the format to WAV.
3. Use Audacity for editing.
4. Set the project rate to 16kHz.
5. Resample the entire recording to match the 16kHz sample rate.
6. Select one second of audio.
7. Export each second of audio as a WAV file. Use 32-bit float as the encoding, matching the bit depth of other samples.
8. The file name can be chosen according to one's own preferences.
9. Create a new folder and name it after your keyword (e.g., "Hello").
10. Move all the samples to this folder.
11. Place the folder alongside other sample keyword folders.
    - prevent bias

Aim for a Balanced Dataset to Avoid Bias

Note that the aim should be set for a balanced dataset. For example a dataset where each class or category of interest has approximately the same number of instances. This is particularly important when training a model, as an imbalanced dataset may lead the model to be biased toward the majority class, potentially hindering its ability to generalize well to minority classes.

For example, if you're training a model to recognize different accents, having a balanced dataset would mean ensuring that you have a similar number of examples for each accent you want the model to identify. This helps prevent the model from becoming overly influenced by the accent with the most instances and allows it to learn patterns from all classes more effectively.

9.11.1. How to Install Audacity

Windows:

1. Visit the official Audacity website: `https://www.audacityteam.org/`
2. Navigate to the "Download" section.
3. Click on the Windows logo to download the installer for Windows.

4. Run the downloaded installer file.
5. Follow the on-screen instructions to complete the installation.

macOS:

1. Visit the official Audacity website: `https://www.audacityteam.org/`
2. Navigate to the "Download" section.
3. Click on the Apple logo to download the installer for macOS.
4. Open the downloaded DMG file.
5. Drag the Audacity application to your Applications folder.

After installation, you can launch Audacity and start using it for audio editing.

9.11.2. License of Audacity Software Product

Audacity is released under the GNU General Public License (GPL). Specifically, Audacity is licensed under the GNU GPL version 2 or later. The GPL is a free software license that allows users to run, study, modify, and distribute the software and its source code.

9.12. How to Train a more Robust Model

- Add more background noise situations
- More different accents
- More different vocal ranges
- More different timbres (tones)
  - Keep in mind that air acts as a dispersive medium, introducing variations in the timbre (tone) of sounds produced by individuals singing at the same frequency (pitch). For instance, a note played on a piano exhibits a distinct timbre compared to the same note played on a guitar.
- Different recording devices
- Variable speed
- Data augmentation techniques
  - e.g., Pitch shifting

  - e.g., Time stretching

- Different ages
  - For instance, the voices of elderly individuals often exhibit tremors.

Table 9.1.: Number of Utterances for Each Word [War18]

Index	Word	Number of Utterances
1	Backward	1,664
2	Bed	2,014
3	Bird	2,064
4	Cat	2,031
5	Dog	2,128
6	Down	3,917
7	Eight	3,787
8	Five	4,052
9	Follow	1,579
10	Forward	1,557
11	Four	3,728
12	Go	3,880
13	Happy	2,054
14	House	2,113
15	Learn	1,575
16	Left	3,801
17	Marvin	2,100
18	Nine	3,934
19	No	3,941
20	Off	3,745
21	On	3,845
22	One	3,890
23	Right	3,778
24	Seven	3,998
25	Sheila	2,022
26	Six	3,860
27	Stop	3,872
28	Three	3,727
29	Tree	1,759
30	Two	3,880
31	Up	3,723
32	Visual	1,592
33	Wow	2,123
34	Yes	4,044
35	Zero	4,052

10. Data Transformation and Data Mining in KDD

10.1. Data Transformation

As was discussed in the Section 5.7.2 the raw audio files are converted to spectrograms. Here, more technical aspects of the conversion are discussed here.

10.1.1. How Does Feature Generation Function?

The method employed mirrors the one utilized throughout Google's production processes, underscoring its substantial practical validation. The theoretical view was discussed in the section 5.7.3. Here, we provide a general overview of its functioning. The procedure initiates by creating a Fourier transform (also referred to as fast Fourier transform or FFT) for a specific time segment—30 ms of audio data in our case. This FFT is generated on data subjected to a Hann window, a bell-shaped function diminishing the impact of samples at the window's ends. The Fourier transform yields complex numbers with real and imaginary components for every frequency. However, our focus is on overall energy, so we sum the squares of the components and apply a square root to obtain magnitudes for each frequency bucket [WS19].

With N samples, a Fourier transform provides information on N/2 frequencies. For a 30-ms duration at a 16,000 samples-per-second rate, 480 samples are required. As our FFT algorithm necessitates a power-of-two input, we pad it with zeros to 512 samples, resulting in 256 frequency buckets. To reduce this, we average adjacent frequencies into 40 downsampled buckets. This downsampling utilizes the mel frequency scale, which is perception-based, assigning more weight to lower frequencies. This process merges higher frequencies into broader buckets, as depicted in Figure 10.1. The output of this process was shown in the Figure 5.7.

A distinctive aspect of this feature generator is its inclusion of a noise reduction step. This involves maintaining a running average for each frequency bucket and subtracting this average from the current value. The goal is to eliminate background noise, which tends to be constant, leaving the speech signals intact. The feature generator retains state to track

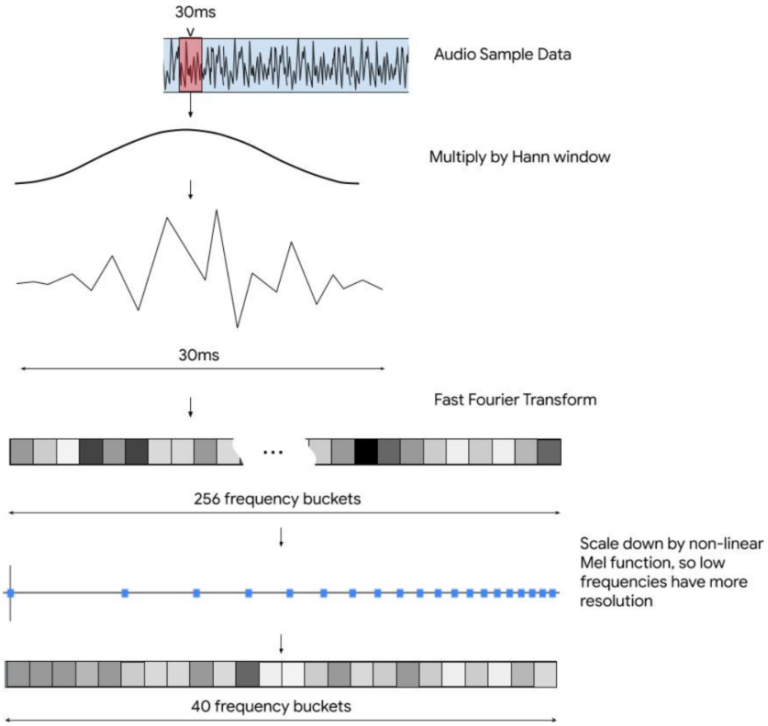


Figure 10.1.: Feature-generation process diagram [WS19].

running averages, necessitating state reset for consistent spectrogram reproduction.

Surprisingly, the noise reduction involves different coefficients for odd and even frequency buckets, creating comb-tooth patterns in the final feature images. Initially perceived as a bug, it was intentionally added to enhance performance. Empirical testing confirmed its impact on accuracy.

Per-channel amplitude normalization (PCAN) auto-gain is applied to boost the signal based on running average noise. Finally, a log scale is employed for all bucket values, preventing loud frequencies from overpowering quieter portions—a normalization aiding the subsequent model's interaction with features.

This entire process is iterated 49 times, employing a 30-ms window moved forward by 20 ms between iterations, covering one second of audio input data. The result is a 2D array of values, 40 elements wide (one for each frequency bucket) and 49 rows high (one for each time slice).

10.2. Data Mining and the Model

The neural network model we utilize is delineated as a concise graph of operations, with Figure 10.2 providing a visual representation of the model structure. This model encompasses a convolutional layer, succeeded

by a fully connected layer, and concludes with a softmax layer. While the convolutional layer is denoted as "DepthwiseConv2D" in the figure, this nomenclature is a quirk of the TensorFlow Lite converter. It is revealed that a convolutional layer with a single-channel input image can be expressed as a depthwise convolution. Additionally, a layer labeled "Reshape_1" serves as an input placeholder rather than an actual operation [WS19].

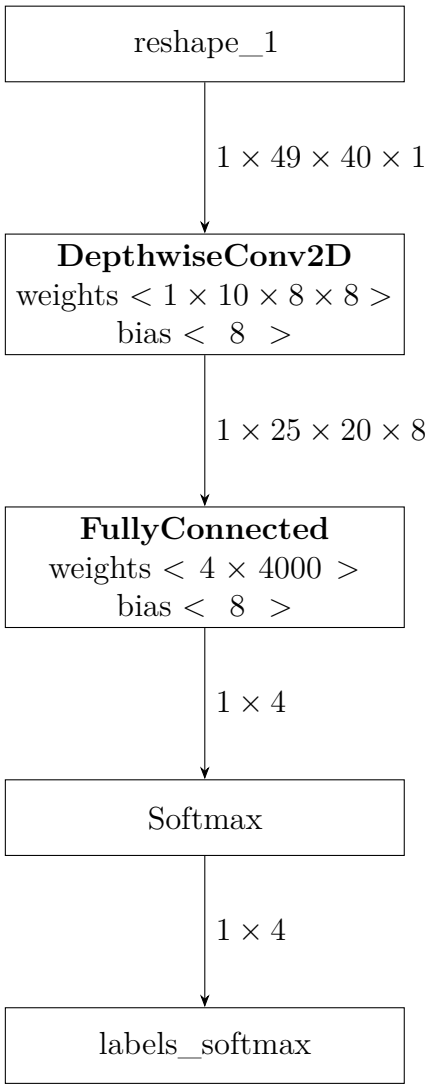


Figure 10.2.: Visual representation of the speech recognition model's graph

Convolutional layers are employed to identify 2D patterns in input images. Each filter, a rectangular array of values, functions as a sliding window across the input, generating an output image that reflects the similarity between the input and filter at each point. The convolution operation can be conceptualized as moving rectangular filters across the

image, with each filter's result indicating its similarity to the corresponding image patch. In this instance, each filter is 8 pixels wide and 10 pixels high, with a total of 8 filters. The filters are shown in the Figure 10.3.

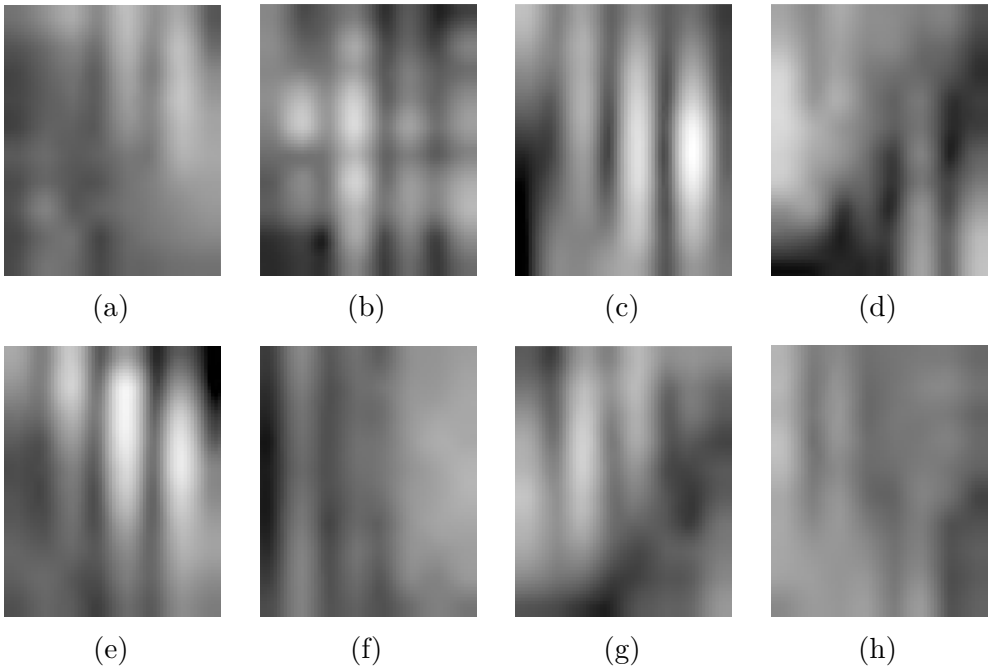


Figure 10.3.: Filter images [Li+21] (a) First filter image (b) Second filter image (c) Third filter image (d) Fourth filter image (e) Fifth filter image (f) Sixth filter image (g) Seventh filter image (h) Eighth filter image

These filters can be viewed as small patches of the input image, aiming to match them with similar sections of the input. High values are written into the corresponding part of the output image where the input resembles the patch. Each filter represents a pattern learned by the model during training to distinguish between different classes. With eight filters, eight distinct output images are produced, each corresponding to the match values of the respective filter as it traverses the input. These outputs are consolidated into a single output image with eight channels.

The subsequent operation involves a fully connected layer, which employs a different pattern matching approach. Instead of sliding a small window, there is a weight for every value in the input tensor, providing an indication of how closely the input aligns with the weights. This is a global pattern matching process, with each class having its own weights, generating four output values for classes like "silence," "unknown," "yes," and "no." The input tensor contains 4,000 values (25 * 20 * 8), representing each class with 4,000 weights.

The final layer is a softmax (explained in the section 5.8.2), enhancing the distinction between the highest output and its competitors. While

the term "probability" is informally used, it requires additional calibration for reliable application. The distribution of training data influences the raw scores, and without proper calibration, it may not accurately reflect the likelihood of certain words.

In addition to these primary layers, biases are incorporated into the results of the fully connected and convolutional layers to fine-tune their outputs. A rectified linear unit (ReLU) activation function follows each layer, ensuring no output is below zero and expediting the convergence of the training process.

10.3. Conclusion

Identifying spoken words efficiently with limited memory space poses a challenging real-world task, demanding engagement with a more extensive set of components compared to simpler instances. The landscape of production machine learning applications necessitates addressing complexities such as feature generation, selecting model architectures, data augmentation, identifying optimal training data, and devising strategies to translate model outcomes into actionable insights. Depending on the product's specific needs, various trade-offs come into play [WS19]. The next step is the transition from the development phase to deployment.

11. Deployment

11.1. Manual Deployment

11.1.1. Setup Arduino IDE (Integrated Development Environment)

Here are the steps to install the Arduino IDE on Windows 11:
Installation

The first step starts by downloading the Arduino software from the official website`https://www.arduino.cc/en/software` . On visiting the link the following prompt appears as shown in figure . Depending on the operating system environment,the necessary software is downloaded and installed.

Figure 11.1.: Arduino Nano website downloads page

In order to carry out the project offline, the software should be downloaded in to the desktop. The latest version of the software is preferred to be downloaded as it contains all the performance improvements and bug fixes.The software can be installed in to multiple operating systems like Windows, Mac OS and Linux. For this project version 1.8.19 was downloaded. After installation the libraries has to be configured for the Arduino Nano BLE 33 sense board to work which is explained in the next section.

Configuration

After installation of Arduino IDE, a window as is shown in figure 11.3

Figure 11.2.: Arduino Nano software version number

In this project an Arduino Nano BLE 33 sense is used, that specific board has to be installed. This done by toggling through option navigating to Tools - > Boards - > Boards Manager as shown in the figure 11.4

On opening the boards manager, a search bar is present that lets you to search, download and install the necessary libraries. A compatible board has to be downloaded and installed for the hardware to work properly. In this scenario, an Arduino Nano mbed OS nano board has to be downloaded and installed as shown in figure 11.5

After installing the boards, the board has to be selected from the menu as shown in figure 11.6. For the keyword spotting project, Arduino Nano 33 BLE has to be selected as the board.

Connect the board to a computer using a USB cable and select the correct port from the menu by going to Tools -> Port -> Arduino Nano BLE 33 as shown in figure 11.7 . If the board is not seen in the drop down menu, ensure that the board libraries are downloaded and installed.

In order to confirm the board is connected and to check the details of the connected board, navigate to Tools -> Get Board Info where a window is produced as shown in figure 11.8

For the Arduino Nano BLE 33 sense to work necessary libraries has to be installed in order to function properly. This can be done my navigating through Tools -> Manage Libraries as shown in figure 11.9. Search for Tensor flow libraries and then download and install them.

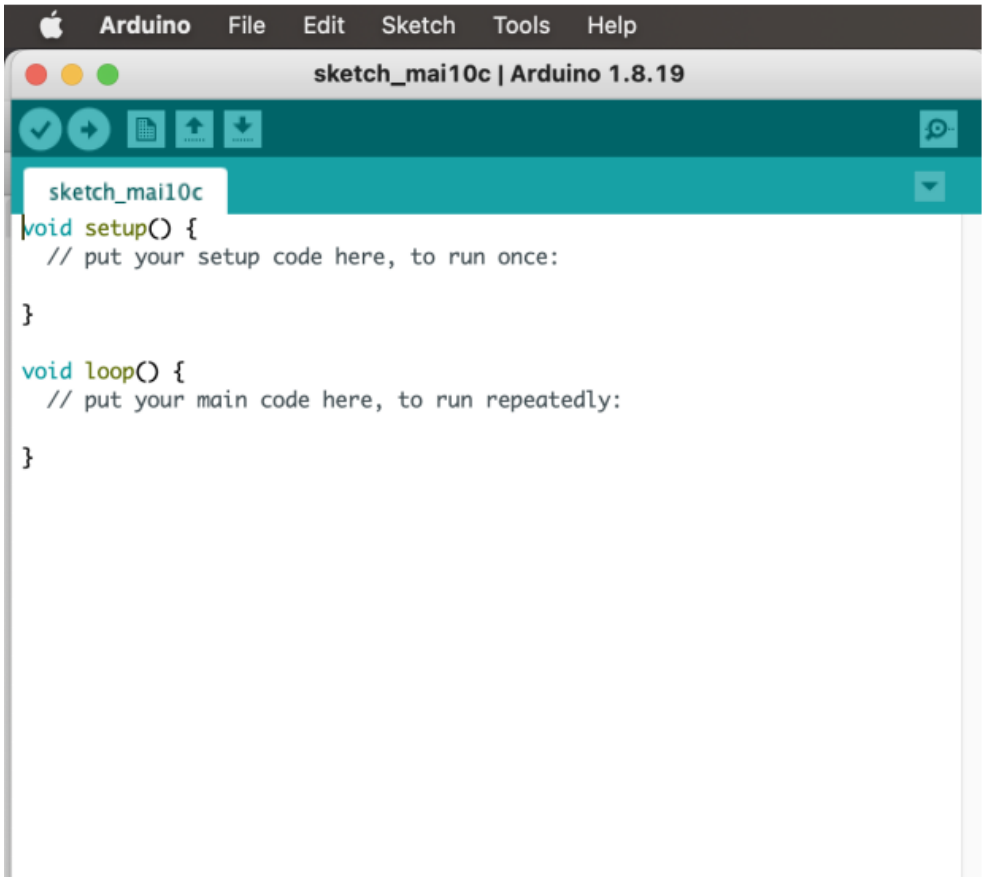


Figure 11.3.: Arduino IDE initial software window

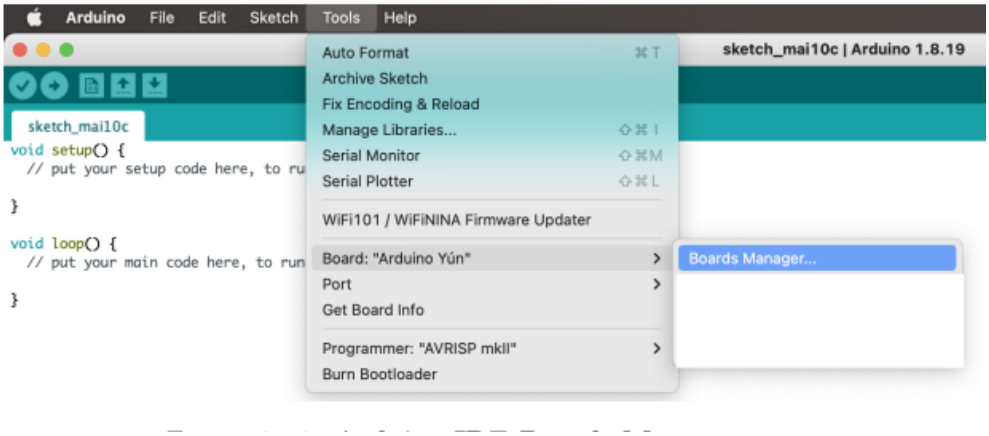


Figure 11.4.: Arduino IDE Boards Manager menu

11.1.2. Connecting an Arduino Nano 33 BLE Sense to a computer

Requirements:

Figure 11.5.: Arduino IDE Boards Manager list

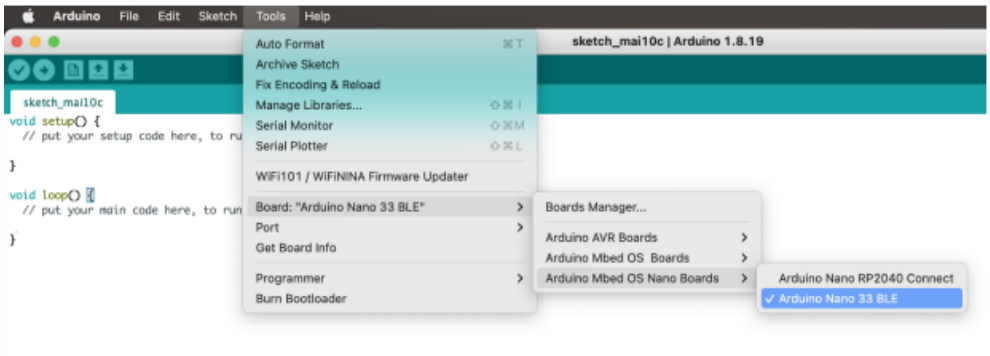


Figure 11.6.: Selection of correct board for the program

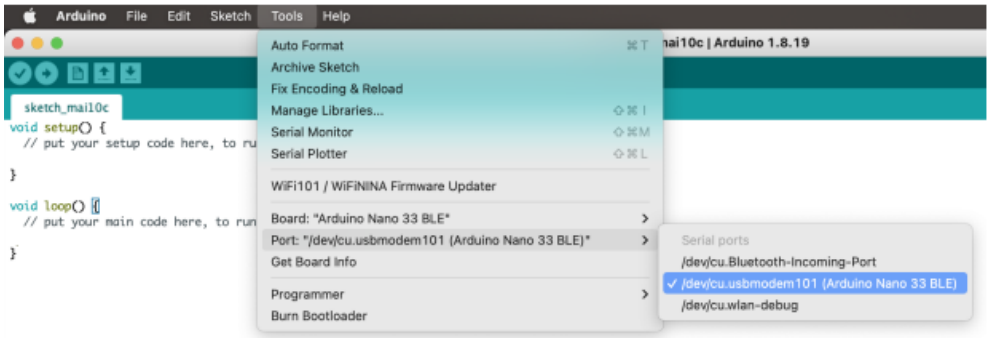


Figure 11.7.: Selection of correct port for uploading the program

1. Arduino Nano 33 BLE Sense.
2. USB Cable.
3. Arduino IDE.

Steps:

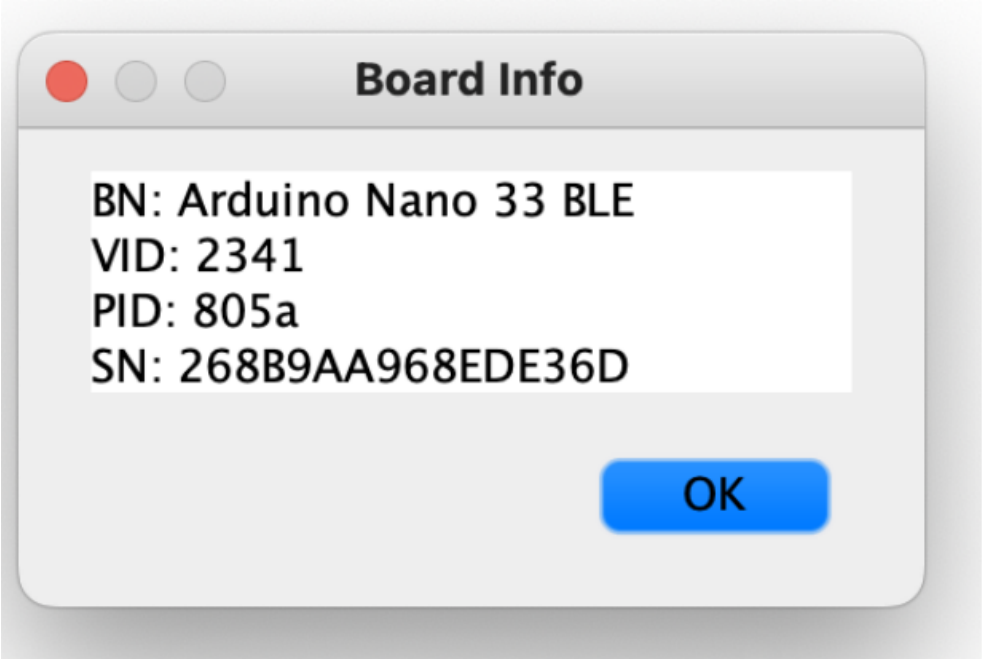


Figure 11.8.: Arduino Board Info

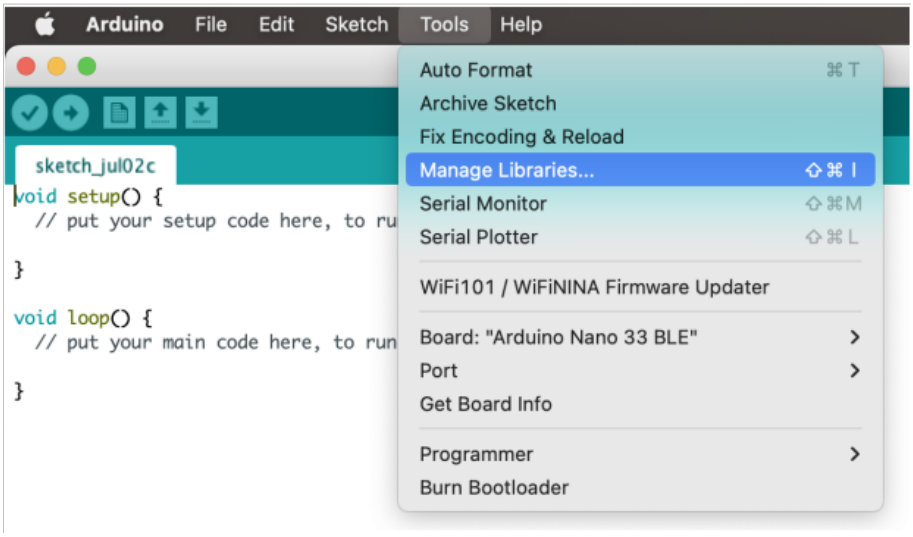


Figure 11.9.: Managing libraries in the Arduino IDE

1. Install Arduino IDE Drivers:
    - Open the Arduino IDE.

- Go to "Tools" > "Board" > "Boards Manager."
- Search for "Arduino mbed-enabled Boards" and install it.

2. Select Arduino Nano 33 BLE Sense:
    - In the Arduino IDE, go to "Tools" > "Board" and select "Arduino Mbed OS Boards" > "Arduino Nano 33 BLE."

3. Install USB Driver
    - For Windows users, you might need to install additional drivers. Follow the instructions provided on the Arduino website for Windows drivers.

4. Connect Arduino to Computer:
    - Connect the Arduino Nano 33 BLE Sense to your computer using the USB cable.

5. Select Port:
    - In the Arduino IDE, go to "Tools" > "Port" and select the port to which the Arduino Nano is connected. The port might look like COMx (Windows) or /dev/ttyUSBx (Linux) or /dev/cu.usbmodemxxxx (macOS).

6. Upload a Test Sketch:
    - Open an example sketch to test the connection. For example, go to "File" > "Examples" > "01.Basics" > "Blink."
    - Click the "Upload" button (right arrow icon) to upload the sketch to the Arduino. You should see the built-in LED on the Nano 33 BLE Sense blink.

11.1.3. Install the TensorFlow Lite runtime

In this project, all you need from the TensorFlow Lite API is the Interpreter class. So instead of installing the large tensorflow package, we're using the much smaller `tflite_runtime` package. To install this on your Arduino, follow the instructions in the Python quick start.

You can install the TFLite runtime using this script.

```
sh setup.sh
```

11.2. TensorFlow Lite

TensorFlow Lite is a mobile library for deploying machine learning models on mobile, microcontrollers, and other edge devices `https://www.tensorflow.org/lite/api_docs`. It is a solution for running machine learning models on mobile devices where luxuries such as huge disk space and GPUs are not usable. TensorFlow Lite is specially optimized for on-device machine learning (Edge ML). It supports platforms such as embedded Linux, Android, iOS, and MCU. TensorFlow Lite takes existing models and converts them into an optimized version within the sort of .tflite file. The new file format supported by TensorFlow Lite is Flat Buffers.

The TensorFlow Lite library provides a set of tools that enables on-device machine learning by allowing developers to run their trained models on mobile, embedded, and IoT devices and computers. It allows one to execute machine learning models easily on a smartphone, allowing one to perform traditional machine learning tasks without the need for an external API or server. In contrast to server-based architectures, TensorFlow Lite is a more effective alternative to mobile model enablement. On mobile devices, it allows offline inference.

TensorFlow Lite is used to develop TensorFlow models and integrate that model into a mobile environment[Boe09]. It is also used to recognize speech keywords "yes" and "no" using the Arduino Nano 33 BLE Sensor. The TensorFlow Lite Micro Library is used to preprocess the audio data and recognize the speech keywords.

Please note that TensorFlow Lite does not optimize model size, so mobile devices may require larger storage. In the TensorFlow Lite process, the expense of reliability and optimization is a trade-off with the model's accuracy. As a result, TensorFlow Lite models are less accurate than their full-featured counterparts.

11.2.1. Code Example

To recognize speech keywords "yes" and "no" using the Arduino Nano 33 BLE Sensor, you can use the TensorFlow Lite Micro Library [Ard12].Here is a sample code in Python that you can use as a starting point:

This code uses the micro_speech library to record audio from the microphone and preprocess it for the TensorFlow Lite model. The model is loaded from the file, which should be trained to recognize the speech keywords "yes" and "no". The recognized keyword is printed to the console.

Please note that this is just a sample code and you will need to modify it to suit your specific requirements. You may also need to install additional libraries and dependencies to run this code on your system. Please refer to the given sample code 11.1

11.2.2. Use in the project

Creating a speech keyword recognition model and deploying it on Arduino Nano 33 BLE Sense involves several steps. Due to the resource constraints of the Arduino Nano, we'll use a simple machine learning model. One popular approach for this task is to use MFCC features with a machine learning classifier like Support Vector Machine (SVM). Here's the Python code for our model:

Please refer to the given code - (see Listing 11.2).

Please refer to the given code - (see Listing 11.3).

LED control:

Based on the recognized command, control the onboard LEDs accordingly

- For "yes" command: Illuminate the green LED.
- For "no" command: Illuminate the red LED.
- For unrecognized keyword the blue LED would be turned on.

11.3. TensorFlow Lite Micro

TensorFlow Lite Micro is a framework that provides a set of tools for running neural network inference on microcontrollers `https://www.tensorflow.org/lite/api_docs`. It is designed to run machine learning models on microcontrollers and other devices with only a few kilobytes of memory. The core runtime just fits in 16 KB on an Arm Cortex M3 and can run many basic models. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.

TensorFlow Lite Micro is a solution for running machine learning models on mobile devices where luxuries such as huge disk space and GPUs are not usable. It is specially optimized for on-device machine learning (Edge ML) 2. It supports platforms such as embedded Linux, Android, iOS, and MCU 2. TensorFlow Lite Micro takes existing models and converts them into an optimized version within the sort of .tflite file. The new file format supported by TensorFlow Lite Micro is Flat Buffers.

The TensorFlow Lite Micro library provides a set of tools that enables on-device machine learning by allowing developers to run their trained models on mobile, embedded, and IoT devices and computers 2. It allows one to execute machine learning models easily on a smartphone, allowing one to perform traditional machine learning tasks without the need for an external API or server `https://www.tensorflow.org/lite/api_docs`. In contrast to server-based architectures, TensorFlow Lite Micro is a more effective alternative to mobile model enablement. On mobile devices, it allows offline inference.

TensorFlow Lite Micro is used to develop TensorFlow models and integrate that model into a mobile environment 2. It is also used to recognize speech keywords "yes" and "no" using the Arduino Nano 33 BLE Sensor. The TensorFlow Lite Micro Library is used to preprocess the audio data and recognize the speech keywords.

Please note that TensorFlow Lite Micro is limited to model inference and does not support training 2. It is also important to note that TensorFlow Lite does not optimize model size, so mobile devices may require larger storage. In the TensorFlow Lite process, the expense of reliability and optimization is a trade-off with the model's accuracy. As a result, TensorFlow Lite models are less accurate than their full-featured counterparts.

```
##############################
# Original Author: TensorFlow team
# Source: https://www.tensorflow.org/lite/api_docs/python/tf/
    lite
# Added to the project by: Malik Al Ashter Ghansletwala
# Date added: 15.12.2023
# Path: ML23-01-Keyword-Spotting-with-an-Arduino-Nano-33-BLE-
    Sense\report\Code\Deployment\TensorflowLiteSampleCode.py
# Version: 2
# Reviewed by:
# Review Date:
##############################

import tensorflow as tf
import numpy as np
import micro_speech

# Load the TensorFlow Lite model.
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Initialize the microphone.
microphone = micro_speech.Microphone()

# Start the microphone recording.
microphone.start()

# Record audio for 1 second.
audio = microphone.record(1)

# Preprocess the audio data.
input_data = np.array(audio, dtype=np.float32).reshape(1,
    1960)
input_data = (input_data - 128.0) / 128.0

# Run inference on the model.
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
output_data = interpreter.get_tensor(output_details[0]['index
    '])

# Print the recognized keyword.
if output_data[0][0] > output_data[0][1]:
    print("Keyword: on")
else:
    print("Keyword: off")
```

Code/Deployment/TensorflowLiteSampleCode.py

Listing 11.1.: Sample code

```
##############################
# Author: Malik Al Ashter Ghansletwala
# Date added: 15.12.2023
# Path: ML23-01-Keyword-Spotting-with-an-Arduino-Nano-33-BLE-
    Sense\report\Code\Deployment\DeployableCode.py
# Version: 2
# Reviewed by:
# Review Date:
##############################
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import librosa
import os

def extract_mfcc(audio_path, max_pad_len=100):
    audio, sample_rate = librosa.load(audio_path, res_type='
    kaiser_fast')
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate,
    n_mfcc=40)
    pad_width = max_pad_len - mfccs.shape[1]
    mfccs = np.pad(mfccs, pad_width=((0, 0), (0, pad_width)),
    mode='constant')
    return mfccs

data_dir = 'path/to/training_data'
keywords = ['on', 'off', 'stop', 'go']

X, y = [], []

for keyword in keywords:
    keyword_dir = os.path.join(data_dir, keyword)
    for filename in os.listdir(keyword_dir):
        audio_path = os.path.join(keyword_dir, filename)
        mfccs = extract_mfcc(audio_path)
        X.append(mfccs.flatten())
        y.append(keyword)

X = np.array(X)
y = np.array(y)

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

classifier = SVC(kernel='linear', random_state=42)
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)
```

Code/Deployment/DeployableCode.py

Listing 11.2.: Deployable Code Part-I

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")

# Save the trained model (for deployment on Arduino Nano)
import joblib
joblib.dump(classifier, 'keyword_recognition_model.joblib')
```

Code/Deployment/DeployableCode.py

Listing 11.3.: Deployable Code Part-II

Part V.

Requirements

12. Bill of Materials

12.1. Hardware Bill of Material

The Table 12.1 show the hardware requirements for this project. The Table 12.2 shows the supplements which can be changed based on the preference of one. All of the materials on the list can be purchased through the reichelt website. As can be seen from these tables, a laptop or desktop computer equipped with a USB port is essential as it serves as the primary programming environment. It is where programming, editing, and compiling of the embedded device's programs take place. The connection between the main computer and the embedded device is established via the USB port. It is worth noting that the main computer can run on various operating systems such as Windows, Linux, or macOS [WS19].

Hardware	**Image**	**Quantity**	**Link**	**Price**
Board Arduino Nano 33 BLE Sense		1	reichelt	36,60 €
GOOBAY 93922 USB 2.0 Cable, A male to micro B male, 0.60 m		1	reichelt	2,30 €
FONTASTIC 262425 Power-bank, Li-Po, 10000 mAh, USB / USB-C		1	reichelt	16,99 €

Table 12.1.: Hardware Bill of Materials

The Board Arduino Nano 33 BLE Sense is powered through a power-bank. Additionally, there's no need for an external microphone since the board comes equipped with a built-in microphone.

Hardware	Image	Quantity	Link	Price
ACER ABKEV.001 Laptop		1	reichelt	442,95 €
MANHATTAN 177658 Mouse		1	reichelt	3,95 €

Table 12.2.: Supplements

12.2. Software Bill of Material (List of Packages and Tools)

12.2.1. Tools

The table (Table 12.3) outlines the key specifications of the required programs for this project. It provides essential information including the software name, version, supported operating systems, license type, and a direct link for convenient access. The listed programs, Arduino IDE and PyCharm Community Edition, are integral to the development environment and cater to various operating systems, offering open-source licenses for accessibility and collaborative development. The provided links facilitate swift downloads, ensuring seamless integration into the project workflow.

Software	Version	Operating System	License	Link
Arduino Integrated Development Environment	2.2.1	Windows, macOS, Linux	Open Source (GPL v2)	Arduino IDE
TensorFlow library for Arduino Integrated Development Environment	-	Windows, macOS, Linux	Open Source (Apache v2)	TensorFlow library
PyCharm Community Edition	2022.3.3	Windows, macOS, Linux	Open Source (Apache 2.0)	PyCharm
xxd for Windows	1.11	Windows	Open Source	xxd for Windows

Table 12.3.: Required Tools

TensorFlow

TensorFlow is Google's open-source machine learning library, initially developed internally at Google and released to the public in 2015. It has since gained a substantial external community of contributors and is compatible with Linux, Windows, and macOS platforms. TensorFlow is the primary machine learning library used within Google, and its core code is consistent across both internal and public versions. While it provides a wide array of tools and optimizations for training and deploying models in the cloud, it was originally designed with a focus on desktop environments, prioritizing lower latency and added functionality, which might not be ideal for platforms with limited resources like Android and iPhone devices, where even a few megabytes added to an app's size can significantly impact downloads and user satisfaction. Building TensorFlow for these platforms typically increases the application size, and despite efforts, it remains relatively large, usually not shrinking below 2 MB even with optimization [WS19].

Installation Procedure

The official TensorFlow Lite Micro library for Arduino is available in the tflite-micro-arduino-examples GitHub repository. To install the latest in-development version of this library, you can directly use the GitHub repository. This involves cloning the repository into the folder that contains libraries for the Arduino IDE. The location of this folder may vary by operating system, typically residing in `~/Arduino/libraries` on Linux, `~/Documents/Arduino/libraries/` on MacOS, and `My Docu-`

`ments\Arduino\Libraries` on Windows.

Once you are in that folder in the terminal, you can obtain the code using the `git` command line tool:

```
git clone https://github.com/tensorflow/tflite-micro-arduino-
examples Arduino_TensorFlowLite
```

To update your cloned repository with the latest code, use the following terminal commands:

```
cd Arduino_TensorFlowLite
git pull
```

Arduino IDE

Arduino IDE, short for Integrated Development Environment, is the official open-source software designed for editing, compiling, and uploading code onto Arduino modules [WS19]. Operating on the Java platform, it offers compatibility with a wide range of Arduino modules and supports both C and C++ programming languages. The Arduino Nano 33 BLE Sense employs this versatile IDE, which is the standard choice for programming all Arduino boards, offering the flexibility to work online and offline. Multiple software versions tailored to specific operating systems, such as macOS, Linux, and Windows, are available. Furthermore, the Arduino community provides an online coding platform that allows users to create and store their sketches in the cloud, and it consistently incorporates the latest IDE features and libraries while accommodating new Arduino boards. For the most recent software packages and information, you can access them through the following link: `https://www.arduino.cc/en/software`arduino.cc/en/software.

xxd for Windows

`xxd` is a hexdump utility that plays a key role in generating a hexadecimal representation of binary data, and in the context of this project, it is employed to facilitate the conversion process. The TFLite model, being a binary file, needs to be represented in a human-readable and manipulable format for certain operations, such as embedding it in C code. `xxd` provides the capability to create a hexdump of the TFLite model, making it easier to embed the model within C source code and ensuring compatibility with the project's requirements for model deployment and integration.

Note: Mac and Linux users don't need this requirement.

PyCharm Community Edition Installation Guide:

1. **Download:**
   - Go to the [PyCharm Community Edition download page](https://www.jetbrains.com/
   - Download the installer for your operating system (Windows, macOS, or Linux).
2. **Installation on Windows:**
   - Run the installer executable.
   - Follow the installation wizard, selecting the default options unless you have specific preferences.
3. **Installation on macOS:**
   - Open the downloaded disk image (.dmg) file.
   - Drag the PyCharm icon to the Applications folder.
4. **Installation on Linux:**
   - Extract the downloaded archive to a location of your choice.
   - Navigate to the extracted folder and run the `bin/pycharm.sh` script.
5. **First Launch:**
   - Launch PyCharm.
   - Choose whether to import settings from a previous installation or start with default settings.
6. **Activation:**
   - Activate PyCharm using a JetBrains account or continue with the free Community Edition.
7. **Usage:**
   - Create a new Python project or open an existing one.
   - Write and run your Python code in the PyCharm editor.

Arduino IDE Installation Guide:

For the installation guide refer to the Subsection 11.1.1.

License

The licenses for the software in Table 12.3 are critical in governing usage and distribution. Arduino IDE (v2.2.1) follows the GNU General Public License version 2 (GPLv2), ensuring open-source freedom for users. PyCharm Community Edition (v2022.3.3) is released under the Apache 2.0 license, providing users the liberty to utilize, modify, and distribute the Python development environment. These licenses foster collaboration and transparency, allowing developers to leverage and enhance these tools for diverse projects. Understanding these licenses is crucial for ethical and compliant use of these development resources.

12.2.2. Packages

The Table 12.4 provides information on key Python packages used in the project, including their versions, licenses, and dependencies. The "Dependencies" column outlines the required dependencies for each package, indicating the version constraints where applicable. For a more detailed list of the packages, refer to the `requirements.txt` file.

Short description

- **Python 3.9:**
  - Python is the programming language in which your project is written. Python 3.9 is a stable and widely used version with numerous improvements and features compared to Python 2. Using the latest version is recommended for compatibility, security, and access to the latest language features.
- **TensorFlow 2.15.0:**
  - TensorFlow is a popular open-source machine learning library. In your project, it's likely used for developing and training machine learning models for keyword spotting. TensorFlow provides tools for building neural networks and processing audio data, which are essential for speech-related tasks.
    - Note that `TensorFlow Lite` is used to facilitate the deployment of machine learning models on edge devices with limited resources. The framework provides tools like the TensorFlow Lite Converter, which optimizes models for memory-constrained devices, and supports quantization to reduce model size and improve execution speed on CPUs [WS19].
- **NumPy 1.26.2:**

Package	Version	License*	Dependencies
Python	2.9	Open Source (PSF)	-
tensorflow	2.15.0	Open Source (Apache 2.0)	numpy (>=1.16.6) gast (=0.3.3) keras-preprocessing (=1.1.0) keras-applications (=1.0.8) tensorboard (=2.6.0) termcolor (>=1.1.0) wrapt (>=1.11.1) absl-py (>=0.7.0) grpcio (>=1.8.6) astunparse (=1.6.3) google-pasta (=0.2.0) h5py (>=2.9.0) opt-einsum (=3.3.0) protobuf (>=3.12.2)
numpy	1.26.2	Open Source (BSD)	-
Matplotlib	3.8.2	Open Source (PSF)	cycler (>=0.10.0) kiwisolver (>=1.0.1) numpy (>=1.15) Pillow (>=6.2.0)
Pandas	2.1.4	Open Source (BSD)	numpy (>=1.16.6) python-dateutil (>=2.7.3) pytz (>=2015.7)
pathlib	2.9	Open Source (PSF)	Python built-in
unittest	2.9	Open Source (PSF)	Python built-in
shutil	2.9	Open Source (PSF)	Python built-in

Table 12.4.: Python packages and dependencies. *PSF: Python Software Foundation, BSD: Berkeley Software Distribution

  - NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. In your project, NumPy may be used for efficient handling and manipulation of numerical data, which is common in machine learning tasks.

- **Matplotlib 3.8.2:**
  - Matplotlib is a plotting library for creating visualizations in Python. In the context of your project, it may be used for visualizing data, training progress, or results. For instance, you might use Matplotlib to create graphs or spectrograms of audio data.

- **unittest:**
  - The `unittest` module is a testing framework that comes with Python. It provides a set of tools for constructing and running tests. It is inspired by the testing frameworks in other programming languages. The `unittest` module is part of the Python Standard Library, and its version is tied to the version of your Python interpreter.
- **pathlib:**
  - The `pathlib` module provides an object-oriented interface for file system paths. It is used to handle paths more effectively and is introduced in Python 3.4 and later. The `pathlib` module is part of the Python Standard Library, and its version is tied to the version of your Python interpreter.
- **shutil:**
  - The `shutil` module is a utility module for high-level file operations. It provides a higher-level interface compared to `os` module functions for file operations like copy, move, and delete. The `shutil` module is part of the Python Standard Library, and its version is tied to the version of your Python interpreter.

Download and install

To install Python 3.9, visit the official Python website at `https://www.python.org/downloads/release/python-390/`, download the installer for your operating system, and execute it. During installation, ensure to select the option to "Add Python 3.9.0 to PATH" for simplified command-line access. Follow the on-screen instructions to complete the installation. Verify the installation by opening a terminal or command prompt and entering the command;

```
python --version
```

You should see "Python 3.9.0."

Once Python is installed, you can proceed to install the required packages using the command line. Open a terminal or command prompt and navigate to your project directory. Then, execute the following command in the same directory that the `requirements.txt` file exists:

```
pip install -r requirements.txt
```

This command reads the package requirements from the `requirements.txt` file and installs them into your Python environment. Ensure that you have the `pip` tool installed and that it corresponds to the Python

version you've installed. To install the packages without using the `requirements.txt` file, one can execute the following command for the `tensorflow` and `numpy` packages for example:

```
pip install tensorflow==2.15.0 numpy==1.26.2
```

Licenses

The licenses for the packages in Table 12.4 are vital in defining usage terms. Python follows the Python Software Foundation (PSF) license, granting freedom to use, modify, and distribute. TensorFlow employs the open-source Apache 2.0 license. NumPy, Pandas, and Matplotlib have open-source BSD and PSF licenses, fostering collaboration. The 'six' library, ensuring Python 2 and 3 compatibility, operates under the MIT license. Understanding these licenses is crucial for ethical and legal software use in development and data science.

12.3. `requirements.txt`

Utilizing Python requirements files is an effective method for managing Python modules. These files are straightforward text documents that store a compilation of the modules and packages essential for your project. By generating a requirements.txt file, you can effortlessly identify and install all the necessary modules manually.

Benefits of Using `requirements.txt`:

1. Facilitates comprehensive tracking of Python modules and packages incorporated into your project.

2. Streamlines the installation process of all required modules on any computer, eliminating the need for manual searches through online documentation or Python package repositories.

3. Enables the seamless installation of dependencies on different computers, ensuring compatibility among them.

4. Simplifies project sharing by allowing others to effortlessly install the specified Python modules listed in your `requirements.txt` file, enabling smooth project execution without complications.

5. Offers a straightforward method for updating or adding Python modules to your project by simply modifying the `requirements.txt` file, eliminating the need to scour through your code for every reference to the outdated module.

12.3.1. Creating a `requirements.txt` File:

Begin by navigating to your Python project directory and creating a new `.txt` document. Ensure it is named `requirements.txt` and save it in the same directory as your `.py` files for the project. At this point, there isn't much more to be done for creating a Python requirements file. However, we will delve into the manual installation of specific packages in the terminal.

Alternatively, you can generate a `requirements.txt` file directly from the command line using the following command:

```
pip freeze > requirements.txt
```

The `pip freeze` command outputs a list of all installed Python modules along with their respective versions.

Adding Modules to Your `requirements.txt` File:

The initial step involves opening the text document and inserting the names of the modules you intend to install. For instance, if you wish to incorporate the TensorFlow library into your project, type "tensorflow" on a new line along with the desired version. Consider the following example for your `requirements.txt` file:

```
tensorflow==2.3.1
uvicorn==0.12.2
fastapi==0.63.0
```

After including all the required modules, save the document.

12.3.2. Installing Python Packages from a `requirements.txt` File

Utilizing the Package Manager Pip:

To install packages from it, we will utilize the package manager `pip`. `Pip` is a versatile utility employed for the installation, upgrade, and uninstallation of Python packages. Additionally, it plays a role in managing Python virtual environments and more.

Installing Python Packages from a `requirements.txt` File:

To initiate `pip`, open a terminal or command prompt and navigate to your Python project directory. Once there, enter the following command:

```
pip install -r requirements.txt
```

This command installs all the modules specified in your Python requirements file into the project environment.

Output:

```
Successfully installed absl-py-1.0.0 astunparse-1.6.3 ...
```

Note: It is advisable to set up a new environment before installing packages with your Python requirements file. If you need to remove a module, use the same command but replace "install" with "uninstall." Similarly, use "upgrade" instead of "install" to update previously installed Python packages. As mentioned earlier, employ the command `pip freeze` to generate a list of the Python modules installed in your environment.

12.3.3. Maintaining a Python Requirements File

Step 1: Output a list of outdated packages with `pip list -outdated`.

Output:

```
Package              Version  Latest  Type
click                7.1.2    8.0.3   wheel
fastapi              0.63.0   0.70.0  wheel
gast                 0.3.3    0.5.3   wheel
h5py                 2.10.0   3.6.0   wheel
numpy                1.18.5   1.21.4  wheel
pip                  20.0.2   21.3.1  wheel
setuptools           44.0.0   59.5.0  wheel
starlette            0.13.6   0.17.1  wheel
tensorflow           2.3.1    2.7.0   wheel
tensorflow-estimator 2.3.0    2.7.0   wheel
uvicorn              0.12.2   0.15.0  wheel
```

Step 2: Upgrade the required package with `pip install -U PackageName`. For example, updating `fastapi`:

```
pip install -U fastapi
```

Output:

```
Successfully installed anyio-3.4.0 fastapi-0.70.0 sniffio-
1.2.0 starlette-0.16.0
```

It is also possible to upgrade everything:

```
pip install -U -r requirements.txt
```

Step 3: Check to see if all tests pass.

Step 4: Run `pip freeze > requirements.txt` to update the Python requirements file.

Step 5: Run `git commit` and `git push` to the production branch.

Freezing all dependencies ensures predictable builds. To check for missing dependencies, use:

```
python -m pip check
```

Output:

```
No broken requirements found.
```

12.3.4. Creating a Python Requirements File After Development

While it is possible to create it manually, it's good practice to use the `pipreqs` module. Install it with:

```
pip install pipreqs
```

Running `pipreqs` in the command line generates a file `requirements.txt` automatically:

```
$ pipreqs /home/project/location
Successfully saved requirements file in /home/project/location/requirements
```

12.3.5. Why You Should Use a Python Requirements File

Create a `requirements.txt` file when starting a new data science project. It's a best practice, particularly in the context of version control.

Using Python requirements files is among Python development best practices. It significantly reduces the need for managing and supervising different libraries. Managing library dependencies from one place makes it more convenient and faster compared to pasting a list of dependency paths into the command line.

GitHub provides automated vulnerability alerts for dependencies in your repository when you upload a `requirements.txt` with your code. It checks for conflicts and alerts the administrator if any are detected, and it can even resolve the vulnerabilities automatically.

12.3.6. Contents of the `requirements.txt` File

In order to ensure a consistent and reproducible development environment, a set of Python dependencies and packages are specified in the `requirements.txt` file. This file can be utilized to create an isolated environment using tools like Conda.

- `absl-py=2.1.0=pypi_0`: Google's Abseil library for Python, version 2.1.0 from the Python Package Index (PyPI).
- `astunparse=1.6.3=pypi_0`: Library to unparse Python abstract syntax trees, version 1.6.3 from PyPI.
- `ca-certificates=2023.12.12=haa95532_0`: SSL certificates required for secure connections.
- `cachetools=5.3.2=pypi_0`: Generic caching library, version 5.3.2 from PyPI.
- `certifi=2024.2.2=pypi_0`: Certificates for SSL/TLS, version 2024.2.2 from PyPI.
- `charset-normalizer=3.3.2=pypi_0`: Library for URL encoding normalization, version 3.3.2 from PyPI.
- `flatbuffers=23.5.26=pypi_0`: Library for efficient serialization of structured data, version 23.5.26 from PyPI.
- `gast=0.5.4=pypi_0`: TensorFlow related package, version 0.5.4 from PyPI.
- `google-auth=2.27.0=pypi_0`: Google Authentication Library, version 2.27.0 from PyPI.
- `google-auth-oauthlib=1.2.0=pypi_0`: Library for integrating Google Authentication into OAuth2, version 1.2.0 from PyPI.
- `google-pasta=0.2.0=pypi_0`: TensorFlow related package, version 0.2.0 from PyPI.
- `grpcio=1.60.1=pypi_0`: High-performance RPC (Remote Procedure Call) framework, version 1.60.1 from PyPI.
- `h5py=3.10.0=pypi_0`: HDF5 support for TensorFlow, version 3.10.0 from PyPI.
- `idna=3.6=pypi_0`: Internationalized Domain Names in Applications (IDNA) library, version 3.6 from PyPI.

- `importlib-metadata=7.0.1=pypi_0`: Library for accessing metadata for Python packages, version 7.0.1 from PyPI.
- `keras=2.15.0=pypi_0`: High-level neural networks API, version 2.15.0 from PyPI.
- `libclang=16.0.6=pypi_0`: Library for interacting with C, C++, and Objective-C code, version 16.0.6 from PyPI.
- `markdown=3.5.2=pypi_0`: Markdown parsing library, version 3.5.2 from PyPI.
- `markupsafe=2.1.5=pypi_0`: Library for XML/HTML markup safe string, version 2.1.5 from PyPI.
- `ml-dtypes=0.2.0=pypi_0`: Machine learning data types library, version 0.2.0 from PyPI.
- `numpy=1.26.3=pypi_0`: Fundamental package for scientific computing with Python, version 1.26.3 from PyPI.
- `oauthlib=3.2.2=pypi_0`: OAuth library, version 3.2.2 from PyPI.
- `openssl=3.0.13=h2bbff1b_0`: OpenSSL cryptographic library, version 3.0.13.
- `opt-einsum=3.3.0=pypi_0`: Optimized contraction of multi-dimensional arrays, version 3.3.0 from PyPI.
- `packaging=23.2=pypi_0`: Core utilities for Python packages, version 23.2 from PyPI.
- `pip=23.3.1=py39haa95532_0`: Package installer for Python.
- `protobuf=4.23.4=pypi_0`: Protocol Buffers, version 4.23.4 from PyPI.
- `pyasn1=0.5.1=pypi_0`: ASN.1 types and codecs library, version 0.5.1 from PyPI.
- `pyasn1-modules=0.3.0=pypi_0`: Collection of ASN.1 modules, version 0.3.0 from PyPI.
- `python=3.9.18=h1aa4202_0`: Python programming language, version 3.9.18.
- `requests=2.31.0=pypi_0`: HTTP library for Python, version 2.31.0 from PyPI.

- `requests-oauthlib=1.3.1=pypi_0`: OAuthlib extension for Requests library, version 1.3.1 from PyPI.
- `rsa=4.9=pypi_0`: RSA public key encryption algorithm, version 4.9 from PyPI.
- `setuptools=68.2.2=py39haa95532_0`: Package development and distribution utilities.
- `six=1.16.0=pypi_0`: Python 2 and 3 compatibility library, version 1.16.0 from PyPI.
- `sqlite=3.41.2=h2bbff1b_0`: Self-contained, serverless, and zero-configuration SQL database engine, version 3.41.2.
- `tensorboard=2.15.1=pypi_0`: TensorFlow's visualization toolkit, version 2.15.1 from PyPI.
- `tensorboard-data-server=0.7.2=pypi_0`: TensorBoard data server, version 0.7.2 from PyPI.
- `tensorflow=2.15.0=pypi_0`: Open-source machine learning library, version 2.15.0 from PyPI.
- `tensorflow-estimator=2.15.0=pypi_0`: TensorFlow's high-level API for distributed training, version 2.15.0 from PyPI.
- `tensorflow-intel=2.15.0=pyp_0`: TensorFlow with Intel optimizations, version 2.15.0 from PyPI.
- `tensorflow-io-gcs-filesystem=0.31.0=pypi_0`: TensorFlow I/O library for Google Cloud Storage filesystem, version 0.31.0 from PyPI.
- `termcolor=2.4.0=pypi_0`: ANSI color formatting for output in terminal, version 2.4.0 from PyPI.
- `typing-extensions=4.9.0=pypi_0`: Type hints for Python, version 4.9.0 from PyPI.
- `tzdata=2023d=h04d1e81_0`: Time zone database, version 2023d.
- `urllib3=2.2.0=pypi_0`: HTTP library for Python, version 2.2.0 from PyPI.
- `vc=14.2=h21ff451_1`: Compiler package for Microsoft Visual Studio, version 14.2.
- `vs2015_runtime=14.27.29016=h5e58377_2`: Visual Studio 2015 redistributable runtime, version 14.27.29016.

- `werkzeug=3.0.1=pypi_0`: WSGI utility library for Python, version 3.0.1 from PyPI.
- `wheel=0.41.2=py39haa95532_0`: Built-package format for Python.
- `wrapt=1.14.1=pypi_0`: Function wrapper library for Python, version 1.14.1 from PyPI.
- `zipp=3.17.0=pypi_0`: Backport of Python 3.10's zipapp module, version 3.17.0 from PyPI.

12.3.7. Best Practices for Using a Python Requirements File

- Always use the command `pip freeze` to generate an up-to-date list of Python modules and packages installed in the virtual environment.
- Only list necessary Python modules and packages in `requirements.txt`.
- Save the `requirements.txt` in the project repository for easy installation by other developers.
- Use `pip install -r requirements.txt` to install all listed Python modules and packages.
- Keep `requirements.txt` up to date for the latest versions of Python modules and packages.

12.4. `environment.yml` Conda Environment Configuration

In this section, the contents of the `environment.yml` file are explained. To understand why and how it is produced, and see the use cases, see Chapter 13.

```
name: KeywordSpottingEnv
```

This sets the name of the Conda environment to "KeywordSpottingEnv."

```
channels:
- defaults
```

Specifies the channels from which to obtain packages. In this case, it includes the default Conda channel.

```
dependencies:
- ca-certificates=2023.12.12=haa95532_0
- openssl=3.0.13=h2bbff1b_0
- pip=23.3.1=py39haa95532_0
- python=3.9.18=h1aa4202_0
- setuptools=68.2.2=py39haa95532_0
- sqlite=3.41.2=h2bbff1b_0
- tzdata=2023d=h04d1e81_0
- vc=14.2=h21ff451_1
- vs2015_runtime=14.27.29016=h5e58377_2
- wheel=0.41.2=py39haa95532_0
```

Specifies the system-level dependencies for the Conda environment, including Python version, OpenSSL, SQLite, and other essential packages.

```
- pip:
- absl-py==2.1.0
- astunparse==1.6.3
- ...  % Other Python packages and their versions
```

Specifies Python packages to be installed using `pip` within the Conda environment. It includes various TensorFlow-related packages and their specific versions.

```
prefix: D:\Users\...\anaconda3\envs\KeywordSpottingEnv
```

Specifies the path where the Conda environment will be created.

13. Development Environment

13.1. Introduction

A development environment refers to the integrated set of tools, software, and resources that facilitate the creation, testing, and deployment of software applications. It provides developers with a structured and efficient workflow, enabling them to write, edit, and debug code while ensuring compatibility and efficiency. Development environments typically include code editors, compilers or interpreters, debuggers, version control systems, and other utilities tailored to the specific programming languages or frameworks used. They aim to streamline the development process, enhance collaboration, and improve productivity by offering features such as code autocompletion, syntax highlighting, code refactoring, and seamless integration with other development tools and services.

13.2. What it is used for

A development environment is primarily used by software developers to create, modify, and maintain software applications. It serves as a centralized workspace where developers can write and edit code, compile or interpret it into executable form, debug and test their applications, and manage their projects. The development environment provides a range of tools and features that streamline the development process, such as code completion, syntax highlighting, error checking, and version control integration. It also allows for efficient collaboration among developers, enabling them to work on the same codebase, share and review code, and track changes. Ultimately, a development environment enhances productivity, facilitates efficient coding practices, and supports the creation of high quality software applications.

13.3. versions

Some of the main packages used are listed here. For the complete list See chapter 12.

1. Programming Language: Python (**version: 3.9.18**)

- The code is written in Python, a widely used programming language for data analysis and machine learning tasks.

2. Libraries and Packages:
   - tensorflow (**version: 2.15.0**): An open-source machine learning library developed by Google. TensorFlow is widely used for deep learning tasks and neural network implementations.
   - numpy (**version: 1.26.3**): Used for numerical computations and operations on arrays. It provides efficient mathematical functions for working with arrays.
   - matplotlib (**version: 3.7.2**): Used for data visualization. It provides functions to create various types of plots and charts.
3. Conda (**version: 23.3.1)**:
   - Conda is an open-source package and environment management system for Python. It simplifies installing, managing, and updating software packages and dependencies.
4. Data Files:
   - The code reads data from several CSV files located in the 'Data' directory. These files include **'stores.csv', 'items.csv', 'transactions.csv', 'oil.csv', 'holidays_events.csv', and 'train.csv'** .

It's important to note that the code assumes the availability of the required libraries, the proper installation of Python, and the presence of the data files in the specified directory.

13.4. Description

A **conda environment** refers to a directory housing a distinct set of conda packages that have been installed. For instance, you might have an environment with NumPy 1.7 and its associated dependencies, while another environment retains NumPy 1.6 for legacy testing purposes. Modifying one environment does not impact the others, ensuring separation. Switching between environments is made simple by activating or deactivating them. Furthermore, sharing your environment involves providing a copy of your environment.yaml file to others [**Conda-environments:2023**].

Why a Conda Environment is chosen

the main differences between a conda environment and the default environment lie in their package management and isolation capabilities. Conda environments provide a comprehensive package management system, allowing easy installation, update, and removal of packages while

handling dependencies and ensuring compatibility. They offer isolation by creating separate environments with their own package sets, enabling conflict-free work on different projects. Conda environments also ensure cross-platform compatibility, allowing consistent behavior across different operating systems. In contrast, the default environment typically has limited package management capabilities and less isolation, potentially leading to conflicts and platform dependencies.

13.4.1. Main Functions

1. **Environment Creation:** The primary function of Conda is to create isolated environments. Each environment can have its own Python version and packages, independent of the system's default Python installation. This allows you to work on multiple projects with different requirements without conflicts.

2. **Package Management:** Conda provides robust package management capabilities. It allows you to install, update, and remove packages within an environment. You can specify the exact versions of packages or let Conda handle the dependencies automatically.

3. **Environment Activation/Deactivation:** Conda allows you to activate and deactivate environments. Activation sets the environment variables and modifies the system's PATH to use the packages installed within the active environment. This ensures that the correct Python interpreter and packages are used when running commands or scripts.

4. **Environment Sharing and Replication:** Conda environments can be exported and shared with others. This allows collaborators or other users to recreate the same environment with the exact set of dependencies needed for a project. It ensures that everyone is using the same versions of packages, reducing compatibility issues.

13.4.2. Subfunctions

1. **Create an Environment:** You can create a new Conda environment using the `conda create` command. For example:

   ```
   conda create --name myenv
   ```

2. **Activate an Environment:** To activate an environment, you can use the `conda activate` command. For example:

   ```
   conda activate myenv
   ```

3. **Deactivate an Environment:** To deactivate an environment, you can use the `conda deactivate` command. For example:

   ```
   conda deactivate
   ```

4. **Install Packages:** You can install packages into an environment using the `conda install` command. For example:

   ```
   conda install numpy
   ```

5. **Update Packages:** You can update packages within an environment using the `conda update` command. For example:

   ```
   conda update numpy
   ```

6. **Remove Packages:** To remove packages from an environment, you can use the `conda remove` command. For example:

   ```
   conda remove numpy
   ```

7. **Export an Environment:** You can export an environment to a YAML file using the `conda env export` command. For example:

   ```
   conda env export --name myenv --file environment.yml
   ```

8. **Create an Environment from a YAML file:** You can create an environment from a YAML file using the `conda env create` command. For example:

   ```
   conda env create --file environment.yml
   ```

These are some of the main functions and subfunctions of a Conda environment. Using these commands and capabilities, one can effectively manage their Python packages and create reproducible environments for their projects.

13.5. Installation

1. **Download Anaconda**: Visit the Anaconda website (`https://www.anaconda.com/download`) and download the appropriate version of Anaconda for your operating system (Windows, macOS, or Linux)..

2. **Run the Installer**: Once the Anaconda installer is downloaded, locate the file and run it. Follow the on-screen instructions to install Anaconda on your system. Choose the default installation options unless you have specific preferences.

3. **Open Anaconda Prompt**: After installation, open the Anaconda Prompt. On Windows, you can find it by searching for "Anaconda Prompt" in the Start menu. On macOS and Linux, you can open a terminal window and activate the conda base environment.

4. **Create a New Environment**: To create a new conda environment, use the following command:

   ```
   conda create --name KeywordSpottingEnv
   ```

   You can choose any name you like by replacing `KeywordSpottingEnv` with the desired name for your environment.

5. **Activate the Environment**: Once the environment is created, activate it using the following command:

   ```
   conda activate KeywordSpottingEnv
   ```

6. **Install Packages**: With the environment activated, you can now install packages into it. For example, to install the named `tensorflow` package version 2.15.0, use the following command:

   ```
   conda install tensorflow=2.15.0
   ```

   Replace `tensorflow` with the name of the package you want to install. You can install multiple packages in one command by separating them with spaces.

   In our case, these packages should be installed:

   ```
   conda install numpy
   conda install matplotlib
   ```

7. **Manage Packages**: You can update packages in your environment using the `conda update` command. For example, to update `numpy`, use:

   ```
   conda update numpy
   ```

   To remove a package, use the `conda remove` command. For example, to remove `numpy`, use:

   ```
   conda remove numpy
   ```

8. **Deactivate the Environment**: When you're done working in the environment, you can deactivate it using the following command:

   ```
   conda deactivate
   ```

   This will return you to the base conda environment.

You can create additional environments and repeat the steps above to manage and work with different sets of packages as per your project requirements.

13.5.1. Configuration

Configure using PyCharm (special Case)

Here, the Configuration of the Python Interpreter in PyCharm to use your Conda environment is explained.

1. In the settings/preferences window, navigate to the "Project" settings and select your project from the list.

2. Under the project settings, locate the "Python Interpreter" option and click on it.

3. In the Python Interpreter settings, click on the gear icon and select "Add..."

4. In the "Add Python Interpreter" dialog, choose the "Conda Environment" option and select the "Existing environment" radio button.

5. Click on the "..." button next to the "Interpreter" field and browse to the location of your Conda environment. The Conda environments are usually stored in a directory named "envs" inside your Anaconda or Miniconda installation directory.

6. Once you have selected the interpreter from the Conda environment, click on the "OK" button.

7. PyCharm will now set up the interpreter and show it in the list of available interpreters. Click "OK" again to save the changes and close the settings/preferences window.

8. Now, when you work on your project in PyCharm, it will use the selected Conda environment as the interpreter. You can install additional packages, run your code, and access all the functionalities provided by the Conda environment.

By configuring the Python Interpreter in PyCharm to use your Conda environment, you ensure that PyCharm uses the correct Python interpreter and the packages installed within that environment.

Share the environment for other team members (general Case)

Once you have created your conda environment, you can produce a YAML file that captures the environment's configuration. This YAML file can then be used to recreate the environment on another system or share it with others.

Thr YAML file defines the dependencies and configurations of the Conda environment, including the required packages and their versions. It can be used to recreate the same environment on another system.

Here's how you can produce the YAML file and configure an environment based on it:

Generating the YAML file:

- Open your terminal or command prompt.
- Activate the conda environment you want to capture using the command: `conda activate <environment_name>`.
- Use the following command to generate the YAML file: `conda env export > environment.yaml`.
- This command exports the environment's configuration, including package names and versions, into a file named `environment.yaml`. You can choose a different name if desired.

Configuring an environment using the YAML file:

- To create a new environment based on the YAML file, make sure you have conda installed and open your terminal or command prompt.
- Navigate to the directory where the `environment.yaml` file is located.
- Run the following command to create the environment: `conda env create -f environment.yaml`.
- Conda will read the YAML file and create a new environment with the same package configuration.

Note: The `environment_name` in the YAML file will be the name of the newly created environment. If you want to specify a different name, you can modify the `name` field in the YAML file before running the `conda env create` command.

what is more special

- **Check permissions:** Before running the script, ensure that the user has appropriate permissions to create folders in the specified directory. Insufficient permissions may result in failure to create the "data" folder. If encountering issues, verify and adjust the write permissions for the target directory.
- Mac and Windows users should be aware of the differences of these operationg systems while working with Python:

1. **File Paths**: The format of file paths can differ between Mac and Windows. Windows typically uses backslashes (\) to separate directories in a file path (e.g., "C:\Program Files\MyFile.txt"), while Mac uses forward slashes (/) (e.g., "/Users/Username/-Documents/MyFile.txt"). When dealing with file paths in your Python code, it's important to use the appropriate path separators based on the target platform.

2. **Line Endings**: Windows and Mac use different characters to represent line endings in text files. Windows uses a combination of carriage return (\r) and line feed (\n) characters ("\r\n"), whereas Mac uses only the line feed character ("\n"). While this distinction is usually handled automatically by text editors and programming environments, it can occasionally cause issues when working with files that have been created on a different platform.

3. **Platform-Specific Libraries**: Some Python libraries may have platform-specific functionality or dependencies. For example, libraries that interact with the underlying operating system, such as accessing specific system resources or executing shell commands, may require different code or have different behavior on Mac and Windows. It's important to consult the documentation of such libraries to understand any platform-specific considerations.

4. **Case Sensitivity**: Mac is a case-sensitive file system by default, whereas Windows is case-insensitive. This can impact the behavior of Python code that relies on file names or other identifiers being matched exactly. It's essential to be consistent with casing when referencing files or variables to ensure cross-platform compatibility.

13.6. Program "Hello World"

This section provides an example program or code snippet to demonstrate the basic functionality of the development environment.

13.6.1. Description and First Steps

The "Hello World" program is a simple introductory program that outputs the phrase "Hello, World!" It serves as a starting point to verify that the development environment is correctly set up and can execute code.

13.6.2. Example manual: Creation and Use of Conda Environments

1. Create a Conda environment using Anaconda Prompt on your system:

   `conda create --name hello_env python=3.9`

   When prompted with "`Proceed ([y]/n)?`", type `y` and press Enter.

   This command creates a new Conda environment named "hello_env" and specifies Python version 3.9.

2. Activate the Conda environment:

   `conda activate hello_env`

   This command activates the "`hello_env`" environment, indicating that we want to use this environment for our Python code.

3. Create a Python file named `HelloWorld.py` and open it in a text editor. Add the following code:

```python
def sayHello():
    """! Prints the message "Hello, World!" to the console.
    @brief Processes This function prints the message "Hello,
    World!" to the console.
    It does not accept any arguments or return any value.
    @return None
    @see sayHello()
    """

    print("Hello, World!")
```

Code/DevelopmentEnvExample/HelloWorld.py

Listing 13.1.: The only functionality of the code is printing `"Hello, World!"`

   This code defines a function `say_hello()` that prints the "Hello, World!" message to the console.

4. Save the `HelloWorld.py` file and close the text editor.

5. If needed, navigate to the directory where your Python file is located using the `cd` command. Execute the Python code:

   `python HelloWorld.py`

```
def main():
    """! The main function of the program.
    @brief Processes This function is the entry point of the
    program. It calls the sayHello() function to print the
    "Hello, World!" message.
    @return None
    @see sayHello()
    """
    sayHello()

# Call the main function to start the program
if __name__ == "__main__":
    main()
```

Code/DevelopmentEnvExample/HelloWorld.py

Listing 13.2.: The function `sayHello()` is called

This command runs the `HelloWorld.py` script using the Python interpreter in the activated Conda environment.

You should see the output `Hello, World!` printed to the console, indicating that the code executed successfully.

6. Deactivate the Conda environment:

   ```
   conda deactivate
   ```

   This command deactivates the current Conda environment.

   The Anaconda Prompt should look like Figure 13.1.

By following these steps, you've created a Conda environment, activated it, written a simple "Hello World" Python code, and executed it within the Conda environment. Conda allows you to manage different environments for different projects, providing isolation and dependency management for your Python applications.

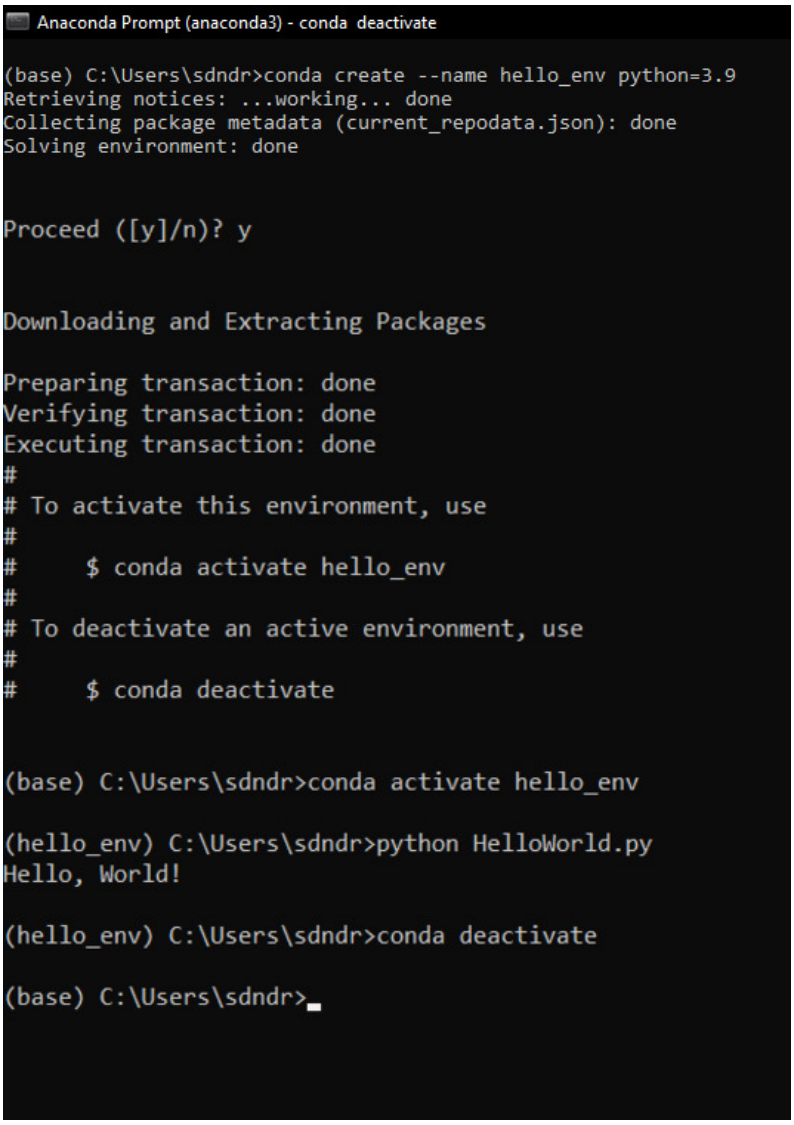


Figure 13.1.: Creating a conda environment for HelloWorld code with Anaconda Prompt

Part VI.

Program

14. Program Flowchart

In this Chapter, a breakdown of each step involved in the keyword spotting program is provided. The program flowchart is shown in Figure 14.1. For a more detailed explanation of the program see Section 15.4 and for understanding the program tests visit chapter 16.

- **Start:** The program initiates the keyword spotting process.
- **Dataset Existence Check:** The program checks if the dataset is already available. If present, it proceeds to the "Load Dataset" step; otherwise, it initiates the process of downloading and extracting the dataset.
- **Load Dataset:** The dataset is loaded from the specified path. A test is conducted to verify the successful loading of the dataset without errors.
- **Audio Preprocessing:** The loaded audio dataset undergoes preprocessing, specifically squeezing the audio to remove unnecessary dimensions. A test validates the correct execution of audio preprocessing steps.
- **Create Spectrogram Dataset:** Spectrograms are generated from the preprocessed audio data, transforming audio signals into frequency representations. The process is tested to confirm spectrogram creation.
- **Build Model:** The neural network model is constructed with specific layers, configurations, and input shapes. A test ensures that the model is built correctly and ready for compilation.
- **Compile Model:** The model is compiled by specifying the optimizer, loss function, and evaluation metrics, preparing it for training.
- **Train Model:** The model is trained on the training dataset, involving shuffling, caching, and prefetching for efficiency. The training process is monitored, and the model's performance is evaluated on the validation dataset.
- **Create Test Dataset:** A separate dataset is created for testing the model's predictions, simulating real-world scenarios.

- **Evaluate Test Predictions:** The model is evaluated on the test dataset to assess its predictive accuracy and generalization to new, unseen data.
- **Save Model:** The trained model is saved to disk for future use or deployment, preserving its architecture and learned parameters.
- **Convert to TFLite and Save:** The saved model is converted into TensorFlow Lite format, a lightweight format suitable for deployment on resource-constrained devices. The converted model is then saved.
- **Test Model Export:** The exporting functions are tested to ensure that the program is capable of exporting the model.
- **Catch and Handle Errors:** The program includes an error-handling mechanism to capture and handle any unexpected errors during the entire process, enhancing resilience to potential issues.
- **Log the Errors:** In case of an error, information about the error is logged. This step aids in debugging and improving the program by providing insights into the nature of encountered issues.
- **End:** The program concludes, marking the successful completion of the keyword spotting pipeline.

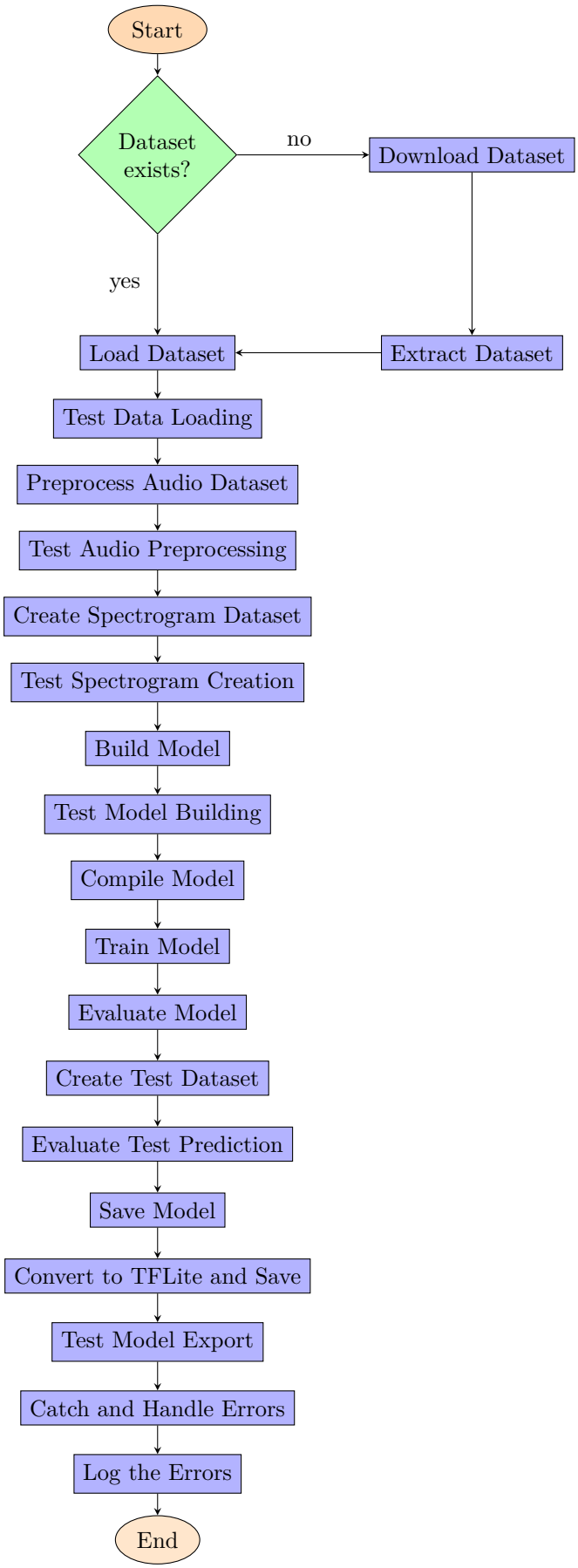


Figure 14.1.: The program flowchart

15. Documentation Development

In software development, the creation of comprehensive technical documentation plays a crucial role in guiding the project through its lifecycle. This documentation serves to address two fundamental questions that shape the entire development process.

Defining the Product: Product Documentation

The first question revolves around determining the nature of the product. What features should the product possess to meet the needs of its users? This aspect of project management is covered under product documentation. A product is essentially a system designed with a specific set of features intended to assist users in achieving their objectives. These features, known as functional requirements, outline the core capabilities that the software should offer.

For instance, in this project, the product documentation would specify the functional requirements related to identifying and recognizing keywords using the Arduino Nano 33 BLE Sense. These requirements might include the supported keywords, sensitivity levels, and integration with other components.

Constructing the Product: Process Documentation

The second critical question is how to go about building the product. This aspect is addressed in process documentation. Process documentation outlines the methodologies, workflows, and procedures that need to be followed during the development lifecycle. It provides a roadmap for the entire development team, ensuring a standardized and efficient approach to building the software.

Considering our example project "Keyword Spotting with an Arduino Nano 33 BLE Sense," process documentation would detail the steps involved in implementing the keyword spotting functionality on the Arduino platform. This includes tasks such as setting up the Arduino development environment, integrating the necessary libraries, and defining the overall workflow for keyword recognition.

15.1. Defining the Product

- **Keyword Recognition:** The product should be capable of recognizing predefined keywords within audio input signals. These keywords are essential for user-defined commands or triggers.
- **Supported Keywords:** The product should support a defined set of keywords, allowing users to customize and specify the words or phrases they want the system to recognize.
- **LED Indication:** The product should feature LED indicators to visually communicate the recognition status. Specifically, a green LED should signify the recognition of a keyword, a red LED for the absence of a keyword, and a blue LED for instances where the system cannot recognize the spoken keyword.
- **Low Resource Consumption:** The solution should be designed to operate with minimal resource consumption on the Arduino Nano 33 BLE Sense, considering its constraints in terms of memory and processing power.
- **Modularity and Extensibility:** The codebase and functionalities should follow modular programming principles, allowing for easy extension or modification of individual components. This modularity enhances maintainability and facilitates future updates.
- **Documentation:** Comprehensive documentation, including installation instructions, configuration guidelines, and usage details, should be provided to assist users and developers in implementing the solution.
- **Testing and Validation:** The product should undergo rigorous testing to validate its performance under various conditions. Testing should cover aspects such as keyword recognition accuracy, response time, and robustness against noise.

15.2. Flowchart of each Step in Tech Development

The steps in tech development are explained in this section. the flowchart of the process is shown in Figure 15.1

Step 1: Install Python

Install Python on your system. You can download the latest version from the official Python website.

```
# Example for Linux
sudo apt-get update
sudo apt-get install python3
```

Step 2: Install PyCharm

Download and install PyCharm, a popular Python IDE, from the JetBrains website.

Step 3: Set Up Environment

Create and set up a virtual environment for your project using the following commands:

```
# Create a virtual environment
python -m venv venv

# Activate the virtual environment
# Example for Windows
venv\Scripts\activate

# Install dependencies from requirements.txt
pip install -r requirements.txt

# Install developer-specific package
pip install devPackageName
```

Make sure to list all necessary Python packages and their versions in the `requirements.txt` file. See Section 12.3 for more details.

Note: You can also use the **.yml** file for setting up the environment. See Section 12.4. You would also need to install `Anaconda` in this case.

Step 4: Install Arduino IDE

Download and install the Arduino IDE from the official Arduino website.

Step 5: Install TensorFlow Library for Arduino IDE

Follow the instructions provided by the TensorFlow Lite for Microcontrollers documentation to install the library in the Arduino IDE: TensorFlow Lite Micro.

Step 6: Set Environment Variables (Windows)

To work with TensorFlow and Arduino in the Command Prompt, you need to set the necessary environment variables. Follow these steps:

1. Right-click on the Start menu and select "System."
2. Click on "Advanced system settings" on the left.
3. In the System Properties window, click on the "Environment Variables" button.
4. Under "User variables," click "New" to add a new variable.
5. Add the following variables:
    - Variable Name: `PATH`
    - Variable Value: Append the paths to the Python Scripts folder and Arduino IDE executable folder, separated by a semicolon. For example:

      `C:\Python39\Scripts;C:\Program Files (x86)\Arduino`

Step 7: Run Tests

test your data, model, and exporting functionalities with the files `testDataUtils.py`, `testModelUtilst.py`, and `testExportUtils.py` in the `Code/KeywordSpotting` directory.

Step 8: Execute the ML Pipeline

Run the `KeywordSpotting.py` file in the `Code/KeywordSpotting` directory to execute the ML pipeline. Ensure that Python is using the virtual environment created earlier.

```
python KeywordSpotting.py
```

Step 9: Convert model.tflite to a C Header File

Execute the following command to convert the TensorFlow Lite model to a C header file:

```
xxd -i model.tflite > tinyConv.cc
```

Step 10: Integrate C Header in TensorFlow Library

Paste the vector from `tinyConv.cc` into the `micro_features_model.cpp` file within the TensorFlow Library for Arduino IDE.

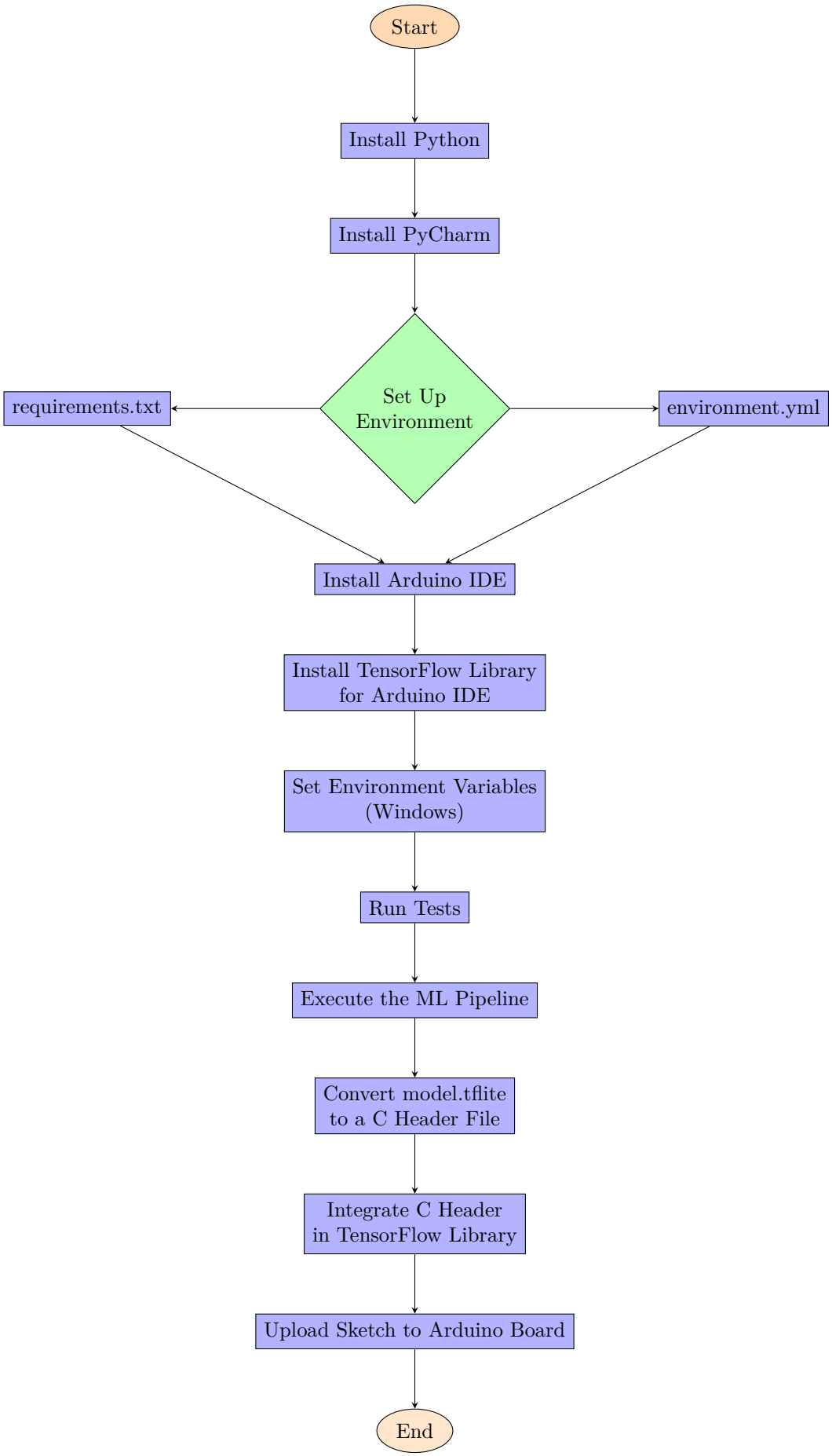


Figure 15.1.: The flowchart of steps in tech development

Step 11: Upload Sketch to Arduino Board

Use the `micro_speech.ino` file to upload the sketch to the connected Arduino board using the Arduino IDE.

Ensure you have the necessary permissions, and the board is properly connected.

To see the programming flowchart, see Chapter 14.

15.3. structure

15.3.1. Modular Programming

Modular programming is a software design approach that decomposes a system into independent modules, such as classes or subsystems. Each module encapsulates a specific functionality with its own implementation. While modules may interact by calling each other's functions or methods, the goal is to minimize dependencies between them. This separation enables developers to work on individual modules without requiring in-depth knowledge of the entire system. The approach aims to manage complexity, with the best modules having implementations that can be modified without affecting other modules. This strategy enhances code maintainability and scalability in large software systems [Ous18].

In this project, emphasis has been placed on the application of modular programming concepts, promoting a more robust and flexible software architecture. To see the programming flowchart, see Chapter 14.

15.3.2. Directory Structure for the Python files

The directory structure itself is a key aspect of modular programming. It organizes related files into logical groups, making it easier to locate and manage different parts of the codebase. It was decided at first that the directory structure for Python files should be as shown below:

```
|-- data/
|-- Doxygen/
|-- envRequirements/
|-- savedModel/
|-- KeywordSpotting.py
|-- Modules/
|   |-- dataUtils.py
|   |-- exportUtils.py
|   |-- modelUtils.py
|-- Tests/
|   |-- testDataUtils.py
```

```
|   |-- testExportUtils.py
|   |-- testModelUtilst.py
|-- handleErrors/
|   |-- errorHandler.py
```

Even though this looks well-organized there are some problems with this approach that are explained in Chapter 16. So the final structure of the code was that all python files be in the same level in a directory. Note that the test files should be named with the convention "`test_`" at the beginning for automation purposes explained in Chapter 16, but due to the agreed convention, the underscore is deleted.

```
|-- data/
|-- Doxygen/
|-- envRequirements/
|-- savedModel/
|-- KeywordSpotting.py
|-- dataUtils.py
|-- exportUtils.py
|-- modelUtils.py
|-- testDataUtils.py
|-- testExportUtils.py
|-- testModelUtils.py
|-- errorHandler.py
```

Folders and module files:

- `dataUtils.py`: Utility functions for dataset loading and preprocessing.
- `exportUtils.py`: Utility functions for model exporting and saving.
- `modelUtils.py`: Utility functions for building and training the model.
- `testExportUtils.py`: Unit tests for exportUtils module.
- `testModelUtilst.py`: Unit tests for modelUtils module.
- `errorHandler.py`: Functions for handling errors across different modules.
- `data` directory is where the data would be downloaded and saved
- `Doxygen` directory is where the documentation files are stored.

- `envRequirements` directory is where the requirement files for creating the required environment are stored. See chapter 13 to understand how to use these files.
- `savedModel` directory is where the model and converted model are saved.

15.3.3. Main Script

`KeywordSpotting.py` serves as the main entry point, coordinating the workflow by leveraging functionalities provided by modularized modules.

15.3.4. Testing

The `testDataUtils.py`, `testExportUtils.py`, and `testModelUtils.py` are dedicated to testing with each test file corresponding to a specific module for targeted and modularized testing. For more information about the testing of these files see Chapter 16.

15.3.5. Error Handling

`errorHandler.py` is a centralized location for error-handling functions, promoting a modular approach to managing unexpected situations. The messages of this module are logged into the `errorHandler.log` file for message handling purposes.

15.3.6. Documented Interface

Doxygen-style comments in each module provide a clear and documented interface, serving as a guide for users and developers.

15.3.7. Separation of Concerns

Each module is responsible for a specific concern, promoting the separation of concerns principle.

15.3.8. Reuse of Components

The modular structure allows for easy reuse of individual modules in other projects.

15.3.9. Ease of Maintenance

The modular structure facilitates maintenance by localizing changes to specific modules.

15.3.10. Testing Independence

The testing modules are independent of each other, allowing for targeted and isolated testing.

15.4. Machine Learning Pipeline

Note: A random seed is set for both TensorFlow (`tf.random.set_seed`) and NumPy (`np.random.seed`). Setting a seed ensures that the random initialization of parameters and any other random processes in the code will be reproducible. This is particularly important when you want to reproduce the same results across different runs of the program.

```
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.1.: Setting the seed for reproducibility

15.4.1. Data Loading

The pipeline begins by loading the dataset using the `loadDataset` function defined in `dataUtils.py`. The dataset consists of audio samples categorized into different commands. If the dataset is not present, it is downloaded and extracted from a predefined URL.

```
    trainDs, valDs = loadDataset(datasetPath, batchSize
=64, validationSplit=0.2, seed=0, outputSequenceLength
=16000)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.2.: Data loading

Load Audio Dataset Function

The `loadDataset` function in Listing 15.3 is designed to load an audio dataset from a specified path. It follows several key steps:

1. **Dataset Directory Setup:** The function converts the given `datasetPath` to a `Path` object using the `pathlib` module: `dataDir = pathlib.Path(datasetPath)`.

2. **Dataset Download:** The function checks if the dataset directory exists. If not, it downloads the dataset from a specified URL using `tf.keras.utils.get_file`.

3. **List Available Commands (Class Labels):** It lists available commands (class labels) in the dataset by filtering out unwanted files like 'README.md' and '.DS_Store'.

4. **Load Audio Dataset:** `tf.keras.utils.audio_dataset_from_directory` is used to load the audio dataset. It automatically splits the dataset into training and validation sets.

5. **Return:** The function returns a tuple containing the training and validation datasets.

6. **Error Handling:** If any exception occurs during the process, it raises an exception using the `errorHandler.errorLoadDataset` function.

15.4.2. Data Cleaning

Data cleaning procedures are detailed in Chapter 9, specifically in sections 9.6 and 9.7. This crucial task is performed by the data provider.

15.4.3. Data Splitting

The `audio_dataset_from_directory` function in TensorFlow facilitates the creation of datasets from audio files organized in a directory structure. The splitting of the dataset into training and validation sets is achieved by specifying the `validation_split` parameter. This parameter designates the fraction of the dataset that will be reserved for validation. Setting `validation_split=0.2` reserves 20% of the data for validation, and the remaining 80% is used for training (See Listings 15.2 and 15.3).

15.4.4. Data Preprocessing

The loaded datasets are further processed using the `preprocessAudioDataset` and `createSpectrogramDataset` functions (See Listings 15.4 and 15.5). The `preprocessAudioDataset` function squeezes the audio data to remove an extra dimension (See Listing 15.4), and the `createSpectrogramDataset` function converts the audio waveforms into spectrograms (See Listing 15.7).

```
def loadDataset(datasetPath, batchSize=64, validationSplit
    =0.2, seed=0, outputSequenceLength=16000):
    try:
        dataDir = pathlib.Path(datasetPath)

        if not dataDir.exists():
            tf.keras.utils.get_file(
                'mini_speech_commands.zip',
                origin="http://storage.googleapis.com/
    download.tensorflow.org/data/mini_speech_commands.zip",
                extract=True,
                cache_dir='.',
                cache_subdir='data'
            )

        else:
            print("The dataset already exists")

        commands = np.array(tf.io.gfile.listdir(str(dataDir))
    )
        commands = commands[(commands != 'README.md') & (
    commands != '.DS_Store')]
        print('Commands:', commands)

        trainDs, valDs = tf.keras.utils.
    audio_dataset_from_directory(
            directory=dataDir,
            batch_size=batchSize,
            validation_split=validationSplit,
            seed=seed,
            output_sequence_length=outputSequenceLength,
            subset='both'
        )

        return trainDs, valDs

    except Exception:
        errorHandler.errorLoadDataset()
```

../Code/KeywordSpotting/dataUtils.py

Listing 15.3.: The `loadDataset` function.

```
        labelNms = trainDs.class_names
        labelNames = np.array(labelNms)
        trainDs = preprocessAudioDataset(trainDs)

        valDs = preprocessAudioDataset(valDs)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.4.: Data preprocessing

```
        trainSpectrogramDs = createSpectrogramDataset(trainDs
    )
        valSpectrogramDs = createSpectrogramDataset(valDs)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.5.: The `createSpectrogramDataset` function converts the audio waveforms into spectrograms

Audio Squeezing

The `preprocessAudioDataset` function is responsible for preprocessing an input audio dataset by squeezing its dimensions. Here are the main steps:

1. **Inner Squeeze Function:** The function defines an inner `squeeze` function, which uses `tf.squeeze` to remove the last dimension of audio data. It returns the squeezed audio and the original labels.
2. **Dataset Mapping:** The `map` function is applied to the input `dataset`, using the `squeeze` function. This is done in parallel using `tf.data.AUTOTUNE` for optimization.
3. **Return:** The preprocessed audio dataset is then returned.
4. **Error Handling:** If an exception occurs during the squeezing process, the function raises an exception using `errorHandler.errorProcessAudio`.

Create Spectrogram Dataset

The `createSpectrogramDataset` function is designed to create a spectrogram dataset from the input audio dataset. The main steps of this function are as follows:

1. **Inner `getSpectrogram` Function:** This function is defined to compute the spectrogram of audio waveforms. It utilizes TensorFlow's signal processing functions, specifically `tf.signal.stft`,

```
def preprocessAudioDataset(dataset):
    def squeeze(audio, labels):
        try:
            audio = tf.squeeze(audio, axis=-1)
            return audio, labels
        except Exception:
            errorHandler.errorProcessAudio()

    dataset = dataset.map(squeeze, tf.data.AUTOTUNE)
    return dataset
```

../Code/KeywordSpotting/dataUtils.py

Listing 15.6.: The `preprocessAudioDataset` function.

to calculate the Short-Time Fourier Transform (STFT) of the audio waveforms. The resulting spectrogram is then processed to obtain the absolute values and expand the last dimension.

2. **Dataset Mapping:** The `map` function is applied to the input `dataset`, using the `getSpectrogram` function. This is done in parallel using `tf.data.AUTOTUNE` for optimization.
3. **Return:** The function returns the spectrogram dataset.
4. **Error Handling:** If an exception occurs during the spectrogram creation process, the function raises an exception using `errorHandler.errorSpectrogram`.

15.4.5. Model Building and Training

The neural network model is constructed using the `buildModel` function shown in Listing 15.8 defined in `modelUtils.py` in Listing 15.9. The architecture includes convolutional and fully connected layers, designed to extract hierarchical features from the spectrogram data. Normalization and dropout layers are incorporated to improve generalization and prevent overfitting.

Adam (short for Adaptive Moment Estimation) is an optimization algorithm. Adam is an extension of the stochastic gradient descent (SGD) optimization algorithm. It combines ideas from two other optimization algorithms: RMSprop (Root Mean Square Propagation) and Momentum.

The model layers are:

- **Input Layer:**
  - Input layer with the defined input shape (size of the spectrograms).

```
def createSpectrogramDataset(dataset):
    def getSpectrogram(waveform):
        try:
            spectrogram = tf.signal.stft(
                waveform, frame_length=255, frame_step=128)
            spectrogram = tf.abs(spectrogram)
            spectrogram = spectrogram[..., tf.newaxis]
            return spectrogram
        except Exception:
            errorHandler.errorSpectrogram()

    dataset = dataset.map(
        map_func=lambda audio, label: (getSpectrogram(audio),
    label),
        num_parallel_calls=tf.data.AUTOTUNE
    )

    return dataset
```

../Code/KeywordSpotting/dataUtils.py

Listing 15.7.: The `createSpectrogramDataset` function.

```
        model = buildModel(inputShape, numLabels, normLayer)
        optimizer = tf.keras.optimizers.Adam()
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.8.: The neural network model is constructed using the `buildModel` function

- **Resizing Layer:**
  - Downsamples the input spectrogram images to a smaller size.
- **Normalization Layer:**
  - Normalizes the spectrogram images using their mean and standard deviation (to be centered around 0 with standard deviation 1).
- **Conv2D Layers:**
  - Apply convolutional operations to capture features.
- **MaxPooling2D Layer:**
  - Reduces spatial dimensions by retaining the maximum values.
- **Dropout Layers:**

  - Introduces regularization by randomly setting a fraction of input units to zero during training.

- **Flatten Layer:**
  - Flattens the 2D output into a 1D array for the fully connected layers.

- **Dense Layers:** The output layer
  - Fully connected layers with specified units and activation functions.

Build Model Function in `modelUtils.py`

The `buildModel` function in `modelUtils.py` is responsible for constructing and compiling a neural network model. The key steps of this function are as follows:

1. **Model Architecture:** The function defines a Keras Sequential model comprising various layers, including convolutional, pooling, dropout, flattening, and dense layers. This architecture is suitable for image classification tasks.

2. **Normalization Layer:** The normalization layer (`normLayer`) is applied to the input data.

3. **Compilation:** The model is compiled using the Adam optimizer, Sparse Categorical Crossentropy loss function, and accuracy as the evaluation metric.

4. **Return:** The compiled Keras Sequential model is returned.

5. **Error Handling:** If an exception occurs during the model construction and compilation, the function raises an exception using `errorHandler.errorBuildModel`.

15.4.6. Evaluating

Evaluation is conducted after the model has been trained on the training dataset and validated on a separate validation dataset. The primary goal is to understand how well the model generalizes to new, unseen audio samples.

```
def buildModel(inputShape, numLabels, normLayer):
    try:
        model = models.Sequential([
            layers.Input(shape=inputShape),
            layers.Resizing(32, 32),
            normLayer,
            layers.Conv2D(32, 3, activation='relu'),
            layers.Conv2D(64, 3, activation='relu'),
            layers.MaxPooling2D(),
            layers.Dropout(0.25),
            layers.Flatten(),
            layers.Dense(128, activation='relu'),
            layers.Dropout(0.5),
            layers.Dense(numLabels),
        ])

        model.compile(
            optimizer=tf.keras.optimizers.Adam(),
            loss=tf.keras.losses.
    SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'],
        )

        return model

    except Exception:
        errorHandler.errorBuildModel()
```

../Code/KeywordSpotting/modelUtils.py

Listing 15.9.: The `buildModel` function.

```
    # Evaluate the model performance
        testDs = valSpectrogramDs.shard(num_shards=2, index
    =0)
        model.evaluate(testDs.cache().prefetch(tf.data.
    AUTOTUNE), return_dict=True)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.10.: Testing and evaluation

Testing and Evaluation Process

Evaluation is initiated by creating a test dataset (`testDs`) from a shard of the validation dataset. This test dataset is then used to evaluate the model's performance using the `evaluate` method (See Listing 15.10).

Here, the `shard` method is employed to create a subset of the validation dataset (`valSpectrogramDs`). The evaluation is performed on this test dataset, which simulates the model's performance on previously unseen

data. The `evaluate` function returns a dictionary containing metrics such as loss and accuracy.

Interpretation of Evaluation Metrics

The evaluation metrics provide valuable insights into how well the model is performing:

1. **Loss**: The loss indicates how well the model is minimizing the difference between predicted and actual labels. A lower loss value is desirable.

2. **Accuracy**: Accuracy represents the proportion of correctly classified samples. It is calculated as the ratio of correctly predicted samples to the total number of samples. Higher accuracy values indicate better model performance.

Early Stopping

In the training phase, an early stopping callback is employed to monitor the model's validation performance during epochs. If the validation performance does not improve over a predefined number of epochs (`patience`), training is halted early (See Listing 15.11). This prevents overfitting and ensures that the model does not memorize the training data without generalizing well to new samples.

```
        EPOCHS = 10
        history = model.fit(
            trainSpectrogramDs.cache().shuffle(10000).
    prefetch(tf.data.AUTOTUNE),
            validation_data=valSpectrogramDs.cache().prefetch
    (tf.data.AUTOTUNE),
            epochs=EPOCHS,
            callbacks=tf.keras.callbacks.EarlyStopping(
    verbose=1, patience=2),
        )
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.11.: Early stopping configuration

15.4.7. Saving

Exporting the Model

The trained model is exported using the `ExportModel` class and saved to a specified directory.

```
    try:
        export = ExportModel(model, labelNms)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.12.

Model Conversion to TensorFlow Lite

The saved model is converted to TensorFlow Lite format using the `convertToTFLite` function. This lightweight model format is suitable for deployment on resource-constrained devices.

```
    try:
        tfliteModelPath = f"{savedModelPath}/model.tflite"
        convertToTFLite(savedModelPath, tfliteModelPath)
```

../Code/KeywordSpotting/KeywordSpotting.py

Listing 15.13.

Save Model Function in `exportUtils.py`

The `saveModel` function is designed to save a TensorFlow model to a specified path using the SavedModel format. The key aspects of this function are as follows:

1. **Parameters:** It takes two parameters: the TensorFlow model (`model`) to be saved and the export path (`exportPath`) where the model should be stored.
2. **Saving Process:** The function utilizes `tf.saved_model.save` to save the model to the specified export path in the SavedModel format. If an error occurs during the saving process, it raises an exception (`errorHandler.errorSaveModel`).

```
def saveModel(model, exportPath):
    try:
        tf.saved_model.save(model, exportPath)
    except Exception:
        errorHandler.errorSaveModel()
```

../Code/KeywordSpotting/exportUtils.py

Listing 15.14.: The `saveModel` function.

15.5. Model Deployment

The command below is a shell command that uses the `xxd` utility to generate a C header file containing the binary representation of the contents of the `model.tflite` file.

```
xxd -i model.tflite > tinyConv.cc
```

`xxd` is a command-line tool available on Unix-like systems (including Linux and macOS) that is used for creating a hex dump of a given file or for converting a binary file to a text format that represents the binary data in hexadecimal. To run the command on a Windows operating system, download xxd for windows from this link: `https://sourceforge.net/projects/xxd-for-windows/` and then run the command in the `Command Prompt`.

The binary contents declared as `alignas(16) const unsigned char g_model[]` in the `microFeaturesModel.cpp` file should be updated with the binary contents declared as `unsigned char model_tflite[]` in the `tinyConv.cc` file. Ensure to replace the existing content in `microFeaturesModel.cpp` with the content of `model_tflite[]` from `tinyConv.cc`. This step is essential for integrating the TensorFlow Lite model into your C or C++ application. For more information see the Chapter 11.

15.6. How to improve

There are several strategies to enhance the current machine learning pipeline's performance and versatility:

Data Augmentation

Implementing data augmentation techniques can artificially increase the size of the training dataset, leading to improved model generalization. Techniques such as random shifts, rotations, and scaling can be applied to the spectrogram data.

Hyperparameter Tuning

Fine-tuning hyperparameters, including learning rates, batch sizes, and the number of layers or units in the neural network, can significantly impact model performance. Employ techniques like grid search or Bayesian optimization to find optimal hyperparameter values.

Transfer Learning

Consider leveraging pre-trained models or transfer learning techniques. Pre-trained models on large audio datasets can capture generic audio features, and fine-tuning on the specific task can expedite convergence and enhance performance.

Ensemble Learning

Combine predictions from multiple models using ensemble learning methods. This approach often leads to improved performance by leveraging the diversity of different models.

Regularization Techniques

Explore regularization techniques such as dropout or L2 regularization to prevent overfitting and improve the model's ability to generalize to unseen data.

Model Architecture Exploration

Experiment with different neural network architectures, including variations in the number and type of layers. Techniques like neural architecture search can automate the exploration process.

Quantization and Pruning

Optimize the model for deployment on resource-constrained devices by applying quantization and pruning techniques. These methods reduce model size and inference time while maintaining acceptable performance.

16. Software Tests

16.1. Introduction

One of the principles of **agile** development (although not exactly being the case for our study) is that testing should be tightly integrated with development, and programmers should write tests for their own code [Ous18].

- unit tests
  - most often written by developers
  - small and focused
  - are often run in conjunction with a test coverage tool[1]
    - Coverage.py
    - pytest-cov

When writing new code or modifying existing code, it is essential to update the corresponding unit tests to ensure continued code functionality and test coverage [Ous18].

Unit tests facilitate refactoring [Ous18]. Without a test suite

- It would be dangerous to make major structural changes to a system.
- bugs will go undetected until the new code is deployed.
  - much more expensive to find and fix

With a good set of tests, developers can be more confident when refactoring because the test suite will find most bugs that are introduced [Ous18].

16.2. "Hello World" Example: How to test Python files

In this section, an example of testing Python files using a simple "Hello World" calculator module is provided. The example consists of two files: `calc.py` (the calculator module) and `testCalc.py` (the unit test module).

[1] ensures that every line of code in the application is tested.

16.2.1. Calculator Module (`calc.py`)

The `calc.py` file contains a basic calculator module with an addition function (see Listing 16.1).

```
def addition(x, y):
    """! Calculate the addition of two numbers.

    This function takes two numbers, ‘x‘ and ‘y‘, and returns
    their sum.

    @param x: The first number.
    @param y: The second number.
    @return: The sum of the two numbers.
    """
    return x + y
```

Code/testExample/calc.py

Listing 16.1.: Calculator Module (`calc.py`)

The `calc.py` module provides a simple `addition` function, allowing users to perform addition operations. This function will be tested in the subsequent unit test module.

16.2.2. Unit Test Module (`testCalc.py`)

The `testCalc.py` file contains unit tests for the functions in the ‘calc‘ module. The primary test case, `TestCalc.test_addition`, checks the correctness of the `addition` function. Below is the relevant portion of the code:

The unit tests include cases with positive integers (5, 5) and mixed integers (-1, 1), with the expected results of 10 and 0, respectively.

16.2.3. Run the Test with `pytest`

To run the tests and for automation purposes, `pytest` package (explained in the section 16.4.1) is used.

Open the command prompt or terminal and navigate to the directory where your project and test files are located.

In the command prompt or terminal, simply run the following command to execute your test:

```
pytest
```

This will discover and run all the tests in your project.

```
import unittest
import calc

class TestCalc(unittest.TestCase):
    """! Unit tests for the calc module."""

    def test_addition(self):
        """! Test the addition function in calc module.

        This test case checks the correctness of the addition
    function
        in the calc module by evaluating it with different
   input values.

        Test cases:
        1. Positive integers (5, 5): Expected result is 10.
        2. Mixed integers (-1, 1): Expected result is 0.
        """
        self.assertEqual(calc.addition(5, 5), 10)
        self.assertEqual(calc.addition(-1, 1), 0)
```

Code/testExample/testCalc.py

Listing 16.2.: Unit Test Module (`testCalc.py`)

Note: To be able to run the pytest command, the name of the file `testCalc.py` should be changed to `test_calc.py` so that pytest would automatically recognize the file as a test file.

The result is shown in Figure 16.1. The output from `pytest` indicates that one test item (test file) was collected.

```
collected 1 item
```

This green line summarizes the test results. It states that one test passed, and it took 0.08 seconds to run.

```
1 passed in 0.08s
```

16.3. Test Files

- The `unittest.main()` block at the end of each script ensures that the unit tests are executed when the script is run as the main program.
- Each script contains specific test cases checking functionalities within their respective modules (`dataUtils`, `modelUtils`, and `exportUtils`).

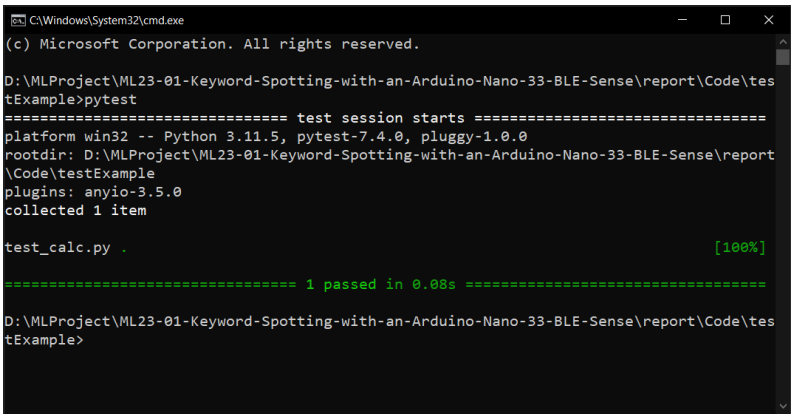


Figure 16.1.: Result of running `pytest` in the direcotry of the test file in `Command Prompt`

16.3.1. `testDataUtils.py`

File Overview

- **Purpose:**
  - This script serves as a unit test suite for the `dataUtils` module.
  - It contains individual test cases to verify the functionality of key functions related to loading, preprocessing, and creating spectrogram datasets from audio data.

Test Cases

`testLoadDataset`

- **Objective:** Checks if the `loadDataset` function successfully loads audio datasets.
- **Testing Approach:**
  - Calls `loadDataset` with specified parameters.
  - Asserts that the returned objects are instances of TensorFlow datasets (`tf.data.Dataset`).

`testPreprocessAudioDataset`

- **Objective:** Checks if the `preprocessAudioDataset` function successfully preprocesses audio datasets.
- **Testing Approach:**
  - Creates a mock dataset.

```
    def testLoadDataset(self):
        """! Test loading dataset function.

        This test case checks if the loadDataset function
    successfully loads audio datasets
        and returns instances of TensorFlow datasets.
        """
        datasetPath = 'data/mini_speech_commands'
        trainDs, valDs = loadDataset(datasetPath, batchSize
    =64, validationSplit=0.2, seed=0, outputSequenceLength
    =16000)
        self.assertIsInstance(trainDs, tf.data.Dataset)
        self.assertIsInstance(valDs, tf.data.Dataset)
```

../Code/KeywordSpotting/testDataUtils.py

Listing 16.3.: The `testLoadDataset` method

```
    def testPreprocessAudioDataset(self):
        """! Test preprocessing audio dataset function.

        This test case checks if the preprocessAudioDataset
    function successfully preprocesses
        audio datasets and returns the preprocessed dataset.
        """
        dataset = tf.data.Dataset.from_tensor_slices(([[1],
    [2]], [3, 4]))
        preprocessed_dataset = preprocessAudioDataset(dataset
    )
```

../Code/KeywordSpotting/testDataUtils.py

Listing 16.4.: The `testPreprocessAudioDataset` method

  - Calls `preprocessAudioDataset` with the mock dataset.

`testCreateSpectrogramDataset`

- **Objective:** Checks if the `createSpectrogramDataset` function successfully creates spectrogram datasets.
- **Testing Approach:**
  - Creates a mock dataset with preprocessed audio data.
  - Calls `createSpectrogramDataset` with the mock dataset.

This test script is a component of a testing strategy, ensuring that the functions in the `dataUtils` module, encapsulated within the `TestDataUtils` class, perform as expected.

```
    def testCreateSpectrogramDataset(self):
        """! Test creating spectrogram dataset function.

        This test case checks if the createSpectrogramDataset
    function successfully creates
        spectrogram datasets from preprocessed audio datasets
    .
        """
        # Example dataset creation with preprocessed data
        audioData = [[1.0, 2.0], [3.0, 4.0]]
        labelData = [3, 4]

        # Preprocess the audio data to ensure it's a 1D
    tensor
        preprocessedAudioData = [tf.convert_to_tensor(
    waveform, dtype=tf.float32) for waveform in audioData]

        # Create the dataset with preprocessed data
        dataset = tf.data.Dataset.from_tensor_slices((
    preprocessedAudioData, labelData))

        # Apply the createSpectrogramDataset function
        spectrogram_dataset = createSpectrogramDataset(
    dataset)
```

../Code/KeywordSpotting/testDataUtils.py

Listing 16.5.: The `testCreateSpectrogramDataset` method

```
class TestDataUtils(unittest.TestCase):
```

../Code/KeywordSpotting/testDataUtils.py

Listing 16.6.: The `TestDataUtils` class

16.3.2. `testModelUtils.py`

File Overview

- **Purpose:**
  - This script serves as a unit test suite for the `modelUtils` module.
  - It includes tests for building a model using the `buildModel` function.

Test Cases

`testBuildModel`

- **Objective:** Checks if the `buildModel` function successfully constructs a model.
- **Testing Approach:**
  - Calls `buildModel` with specified parameters.
  - Asserts that the returned model is an instance of `tf.keras.models.Sequential`.
  - Checks each layer in the model against the expected structure.

This test script is designed to ensure that the `modelUtils` module's `buildModel` function constructs models with the expected structure. The `testBuildModel` function is defined inside the `TestModelUtils` class.

16.3.3. `testExportUtils.py`

File Overview

- **Purpose:**
  - This script serves as a unit test suite for the `exportUtils` module.
  - It includes tests for exporting a model, saving a model, and converting a model to TFLite format.

Test Cases

`TestExportUtils` Class

- **`testExportModel`**
  - **Objective:** Checks if the `ExportModel` class in the `exportUtils` module is functioning correctly.
  - **Testing Approach:**
    - Creates a mock `Sequential` model and label names.
    - Instantiates an `ExportModel` instance and asserts that it is created without errors.
- **`testSaveModel`**
  - **Objective:** Checks if the `saveModel` function in the `exportUtils` module successfully saves a model.
  - **Testing Approach:**
    - Creates a mock `Sequential` model.
    - Calls `saveModel` with the model and a specified save path.
    - Asserts that the save directory is created and, optionally, checks for specific files or conditions within the directory.

```
    def testBuildModel(self):
        """! Test building model function.

        This method tests the functionality of the buildModel
    function in the modelUtils module.
        It checks whether the model is built successfully and
    if the layers match the expected structure.

        @exception AssertionError If the test fails.
        """
        inputShape = (32, 32, 1)
        numLabels = 8  # Adjust the number of labels based on
    your actual model
        normLayer = tf.keras.layers.Normalization(axis=-1)
        model = buildModel(inputShape, numLabels, normLayer)

        self.assertIsInstance(model, tf.keras.models.
    Sequential)

        # Check each layer in the model
        expectedLayers = [
            layers.Resizing(32, 32),
            normLayer,
            layers.Conv2D(32, 3, activation='relu'),
            layers.Conv2D(64, 3, activation='relu'),
            layers.MaxPooling2D(),
            layers.Dropout(0.25),
            layers.Flatten(),
            layers.Dense(128, activation='relu'),
            layers.Dropout(0.5),
            layers.Dense(numLabels)
        ]

        for i, (expected_layer, actual_layer) in enumerate(
    zip(expectedLayers, model.layers)):
            with self.subTest(f"Testing layer {i}"):
                print(f"Actual Layer {i}: {type(actual_layer)
    .__name__}")
                self.assertEqual(type(expected_layer).
    __name__, type(actual_layer).__name__)

                if hasattr(expected_layer, 'input_shape'):
                    self.assertEqual(expected_layer.
    input_shape, actual_layer.input_shape)
                if hasattr(expected_layer, 'units'):
                    self.assertEqual(expected_layer.units,
    actual_layer.units)
                if hasattr(expected_layer, 'activation'):
                    self.assertEqual(expected_layer.
    activation.__name__, actual_layer.activation.__name__)
```

../Code/KeywordSpotting/testModelUtilst.py

Listing 16.7.: The `testBuildModel` method

```
class TestModelUtils(unittest.TestCase):
```

../Code/KeywordSpotting/testModelUtilst.py

Listing 16.8.: The `TestModelUtils` class

* Cleans up by removing the save directory after the test.

- **`testConvertToTFLite`**
  - **Objective:** Checks if the `convertToTFLite` function in the `exportUtils` module handles exceptions during the conversion process.
  - **Testing Approach:**
    * Uses `self.assertRaises` to check if an exception is raised when calling `convertToTFLite` with specified parameters.

```
    def testExportModel(self):
        """! Test ExportModel class.

        This method tests the functionality of the
    ExportModel class in the exportUtils module.
        It checks whether the ExportModel instance is created
     without errors.

        @exception AssertionError If the test fails.
        """
        model = tf.keras.Sequential()
        labelNames = ["label1", "label2"]
        export_model = ExportModel(model, labelNames)
```

../Code/KeywordSpotting/testExportUtils.py

Listing 16.9.: The `testExportModel` method

This test script is part of the testing strategy, ensuring the correct functionality of the `exportUtils` module's key features, encapsulated within the `TestExportUtils` class.

16.4. Automation

To automate the testing process, the pytest package is used.

```
    def testSaveModel(self):
        """! Test saving model function.

        This method tests the functionality of the saveModel
    function in the exportUtils module.
        It checks whether the model is saved successfully and
     the save directory is created.

        @exception AssertionError If the test fails.
        """
        model = tf.keras.Sequential()
        savePath = "savedModelTest"
        saveModel(model, savePath)

        # Check if the savedModelTest directory exists
        self.assertTrue(os.path.exists(savePath))
        # Optionally, you can check for specific files or
    conditions within the savedModelTest directory

        # Clean up: Remove the savedModelTest directory after
     the test
        if os.path.exists(savePath):
            shutil.rmtree(savePath)  # Use shutil.rmtree to
    remove the directory and its contents
```

../Code/KeywordSpotting/testExportUtils.py

Listing 16.10.: The `testSaveModel` method

16.4.1. pytest

Installing `Pytest`

You can install `pytest` using pip:

```
pip install pytest
```

Writing Test Functions

Create a Python file for your tests, and name it with a `test_` prefix (e.g., `test_my_code.py`). Write test functions with names starting with `test_`:

```
# test_myCode.py

def test_addition():
assert 1 + 1 == 2

def test_subtraction():
assert 3 - 1 == 2
```

```
    def testConvertToTFLite(self):
        """! Test converting model to TFLite function.

        This method tests the functionality of the
    convertToTFLite function in the exportUtils module.
        It assumes an error is raised during the conversion
    process.

        @exception AssertionError If the test fails.
        """
        with self.assertRaises(Exception):  # Assuming an
    error is raised in the original code
            convertToTFLite("savedModelTest", "model.tflite")
```

../Code/KeywordSpotting/testExportUtils.py

Listing 16.11.: The `testConvertToTFLite` method

```
class TestExportUtils(unittest.TestCase):
```

../Code/KeywordSpotting/testExportUtils.py

Listing 16.12.: The `TestExportUtils` class

Running Tests

Run pytest from the command line, specifying the test file:

```
pytest test_myCode.py
```

Test Assertions

Use `assert` statements to check conditions. If the condition is False, pytest will raise an exception, and the test will fail:

```
def test_multiply():
result = 2 * 3
assert result == 6, "Multiplication failed"
```

Test Fixtures

Fixtures are a way to set up preconditions for your tests. Use the `@pytest.fixture` decorator:

```
import pytest

@pytest.fixture
def setup_data():
```

```
    data = {"key": "value"}
    return data

def test_data_length(setup_data):
    assert len(setup_data) == 1
```

Running Specific Tests

You can run specific tests by specifying their names:

```
pytest test_myCode.py::test_addition
```

Command Line Options

- `-v` or `-verbose`: Increase verbosity.
- `-k EXPRESSION`: Only run tests with names matching the given substring expression.
- `-cov=PACKAGE`: Measure code coverage.
- `-durations=N`: Print the N slowest tests.

Plugins

pytest has a rich ecosystem of plugins. You can discover and install them to extend pytest's functionality:

```
pip install pytest-someplugin
pytest --someplugin
```

16.4.2. Execution

The result of running `pytest` in the `Command Prompt` in the directory of the test files is shown in Figure 16.2.

```
collected 7 items
```

This line indicates that `pytest` found and collected a total of 7 test items. These items represent individual test files or modules in the project. Note that the name of the files has been changed, starting with "`test_`" so that `pytest` could automatically find them.

```
test_dataUtils.py ...                    [ 42%]
test_exportUtils.py ...                  [ 85%]
test_modelUtilst.py .                    [100%]
```

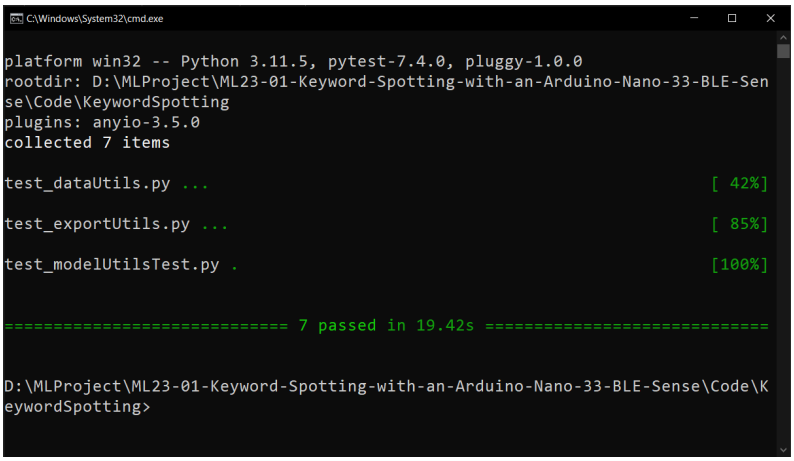


Figure 16.2.: Result of running `pytest` in the direcotry of the test files in `Command Prompt`

Each line corresponds to the progress of test execution for a specific test file. Dots (`...` and `.`) represent successful tests. For the case of `...`, three tests were successful, each dot representing a successful test. The percentage values ([42%], [85%], [100%]) indicate the progress through the entire test suite.

`7 passed in 28.32s`

This final summary line provides an overview of the test results. It states that out of the 7 tests executed, all 7 passed successfully. The total execution time for all tests was 28.32 seconds. The [100%] success rate indicates that all the tests you ran have passed.

Part VII.

Results and Conclusion

17. Results

17.1. Data Transformation

To convert audio to a spectrogram, one-second audio snippets are analyzed in a loop, applying fast Fourier transform (FFT) to each 30-millisecond segment with a 20-millisecond overlap. This process produces a 2D array representing the entire audio sample (See Figure 17.1). This 2D array, commonly known as a spectrogram, captures the intensity of different frequency components across time. The spectrogram is then fed to the CNN model.

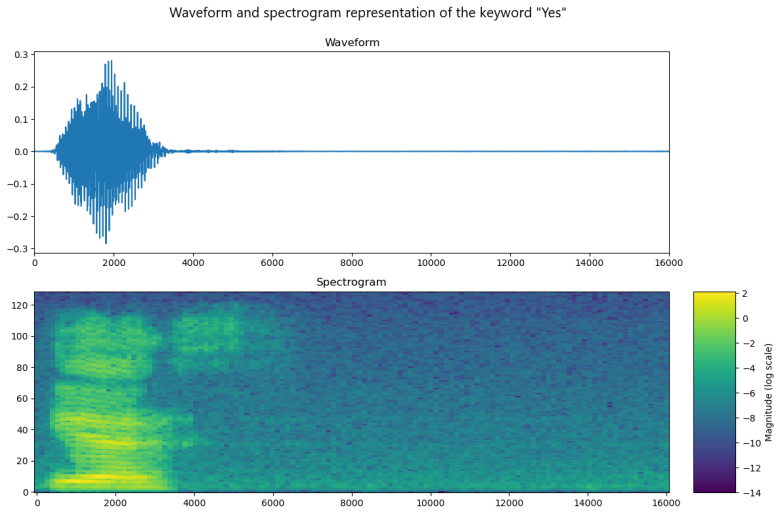


Figure 17.1.: Waveform and spectrogram representation of the keyword "yes"

17.2. Model Export and Conversion

The trained model was successfully exported, and a saved model was created. Furthermore, the saved model was converted to TensorFlow Lite format (TFLite) for deployment on resource-constrained platforms.

17.3. Model Training and Performance

The training process commenced, lasting approximately 22 seconds per epoch. The model exhibited an improvement in performance over the course of the epochs, as evident in the training and validation metrics.

17.4. Epoch-wise Performance

The epoch-wise performance evaluations are shown in Figure 17.2.

- **Epochs 1-5: Initial Training Progress**
  - The initial training phase demonstrates notable progress as the model rapidly enhances its performance.
  - Epoch 1 starts with a loss of 1.7081 and an accuracy of 39.48%. This suggests an initial grasp of data patterns by the model.
  - Subsequent epochs show consistent improvement, reaching a training accuracy of 79.50% by Epoch 5. The corresponding loss decreases from 1.7081 to 0.5908, indicating effective learning.
  - The training time per epoch varies between 7 and 13 seconds, showcasing moderate efficiency.
- **Epochs 6-10: Continued Improvement and Stability**
  - The model continues to refine its features, achieving a peak training accuracy of 89.39% by Epoch 10.
  - The loss further decreases to 0.3241, demonstrating the model's ability to generalize well on the training data.
  - The stability in training time per epoch (7-8 seconds) indicates consistent computational efficiency during this phase.
- **Validation Results after 10 Epochs**
  - The model exhibits strong generalization to unseen data with a validation accuracy of 86.12% after 10 epochs.
  - The validation loss of 0.4351 is slightly lower than the training loss, indicating a good level of generalization without overfitting.
  - The low inference time (2 milliseconds per step) suggests that the trained model is computationally efficient during the testing phase.

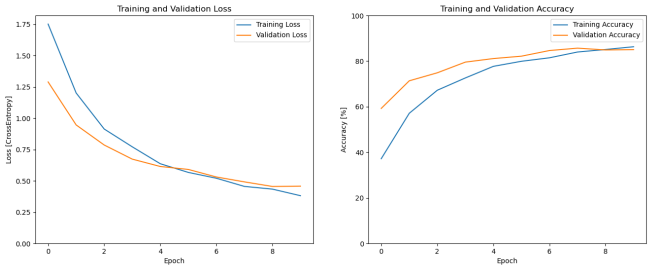


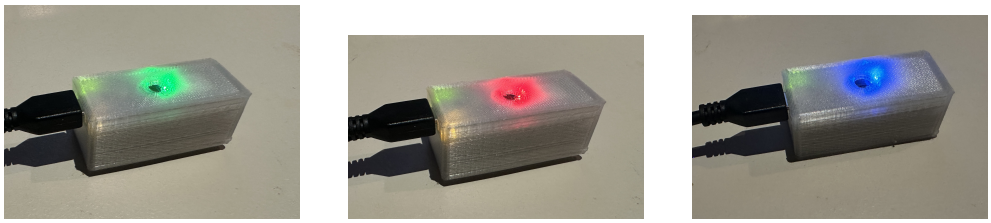
Figure 17.2.: Epoch-wise performance evaluation of the CNN model

17.4.1. Test Dataset Evaluation

The result of this evaluation was a test accuracy of 85%. This indicates that the model correctly classified 85% portion of the samples.

17.5. Arduino Nano 33 BLE Sense Results

The board shows relatively accurate responses when prompted with the designated keyword spoken within a 20cm range. At a distance longer than 20cm, it normally doesn't show a response. However, there are instances where the board may fail to register a response even within a 20cm range. In addition, it may occasionally misinterpret a "yes" or "no" keyword, classifying it as an unknown keyword, which is signified by the activation of a blue LED. Moreover, environmental noise can contribute to the occurrence of unknown keyword responses on the board. While generally reliable, these occasional anomalies highlight the sensitivity of the system to external factors. The board's responses are shown in the Figure 17.3.



(a) (b) (c)

Figure 17.3.: Board response: (a) Response to the keyword "yes" (b) Response to the keyword "no" (c) Response to unknown keyword

18. Conclusion

The report began with a thorough examination of domain knowledge, encompassing TinyML, data collection, preprocessing, model training, deployment, and optimization. The hardware description shed light on the capabilities of the Arduino Nano 33 BLE Sense and its onboard sensors, detailing their functionalities and roles.

The software description offered an overview of the tools and libraries utilized, including the Arduino IDE, TinyML model development tools, and various machine learning frameworks. The integration of Convolutional Neural Networks (CNNs) in the data mining process, particularly for speech recognition applications, represented a strategic adoption of relevant technologies.

The Knowledge Discovery in Databases (KDD) process played a pivotal role, addressing the impracticality of manual data analysis and providing a framework for efficient data mining. The development phase of KDD, from database creation to data transformation, mining, and model evaluation, was systematically discussed. The challenges inherent in data mining were also acknowledged.

The deployment section outlined practical implementation steps, offering a guide for individuals interested in replicating or expanding upon the keyword spotting system. The Bill of Materials, covering both hardware and software components, serves as a comprehensive resource for those looking to explore similar applications.

The combination of hardware, software, and data mining techniques within this context not only addresses the specific challenges of keyword spotting but also contributes to the broader understanding of embedded systems and machine learning integration for real-world applications. The outlined methodologies and insights provide a foundation for future developments in the intersection of edge Artificial Intelligence (AI) and sensor-based technologies.

the keyword spotting system, employing a CNN model and an Arduino Nano 33 BLE Sense, presents a training accuracy of 89.39% and validation accuracy of 86.12%. While the model exhibits proficiency in recognizing keywords from spectrograms, its deployment on the Arduino board reveals occasional sensitivities to environmental factors, leading to sporadic misclassifications and unknown keyword triggers.

18.1. To do

There are additional opportunities for optimizations and enhancements that may be decided upon in the next phase of the project:

1. **Optimization:** Explore further optimization techniques to enhance the model's efficiency and reduce resource utilization on the Arduino Nano 33 BLE Sense.

2. **Expand Keyword Vocabulary:** Investigate methods for expanding the keyword vocabulary recognition capabilities, allowing the system to recognize a broader range of keywords.

3. **Real-world Testing and User Feedback:** Conduct extensive real-world testing to gather user feedback and evaluate the system's performance in diverse environments. This feedback can be invaluable for refining the model and addressing real-world challenges.

4. **Integration of Additional Sensors:** Explore the integration of additional sensors or feature sets to enhance the system's overall contextual awareness and improve keyword recognition accuracy.

5. **User Interface Development:** Develop a user interface (UI) that allows users to interact with and configure the keyword spotting system more intuitively, providing a seamless user experience.

18.2. Future Work

The success of this project lays the groundwork for future advancements in keyword spotting and edge AI applications. Potential avenues for future work include:

1. **Optimizing the Model:** Investigate additional techniques for optimizing the model size without compromising its performance. Explore quantization methods and compression techniques to achieve more efficient model representation.

2. **Enhancing Model Robustness:** Explore methods to improve the model's robustness against variations in input, such as different accents, speaking rates, and background noise. This may include augmenting the training dataset with diverse examples and integrating strategies during model training to enhance robustness.

3. **Iterations in Machine Learning Models:** Explore advanced machine learning models beyond Convolutional Neural Networks to improve keyword recognition on resource-constrained devices.

Experiment with diverse model architectures, including recurrent neural networks and transformer models, to achieve better results.

4. **Multi-modal Integration:** Investigate the integration of multi-modal data, such as combining audio cues with visual or environmental information, to create a more robust and context-aware keyword spotting system.

5. **Dynamic Adaptation:** Develop mechanisms for dynamic adaptation, allowing the system to learn and adapt to changes in the acoustic environment or user preferences over time.

Bibliography

[Ard12] Arduino. *Get Started With Machine Learning on Arduino | Arduino Documentation*. 14/12/2023. URL: https://docs.arduino.cc/tutorials/nano-33-ble-sense/get-started-with-machine-learning.

[BC23] J. Bushur and C. Chen. "Neural Network Exploration for Keyword Spotting on Edge Devices". In: *Future Internet* 15.6 (2023), p. 219. DOI: 10.3390/fi15060219.

[Boe09] G. Boesch. "TensorFlow Lite – Real-Time Computer Vision on Edge Devices (2022)". In: *viso.ai* (21/09/2021). URL: https://viso.ai/edge-ai/tensorflow-lite/.

[C 0] B. P. C. "How to Detect Outliers in Machine Learning – 4 Methods for Outlier Detection". In: *freeCodeCamp.org* (6-07-2022). URL: https://www.freecodecamp.org/news/how-to-detect-outliers-in-machine-learning/.

[DB21] L. Dutta and S. Bharali. "TinyML Meets IoT: A Comprehensive Survey". In: *Internet of Things* 16 (2021), p. 100461. DOI: 10.1016/j.iot.2021.100461.

[DMM20] K. Dokic, M. Martinovic, and D. Mandusic. "Inference Speed and Quantisation of Neural Networks with TensorFlow Lite for Microcontrollers Framework". In: *2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. IEEE. 2020, pp. 1–6. DOI: 10.1109/SEEDA-CECNSM49515.2020.9221846.

[FAD18] M. Fezari and A. Al Dahoud. "Integrated development environment "IDE" for Arduino". In: *WSN applications* (2018), pp. 1–12.

[FPSS96] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. "The KDD Process for Extracting Useful Knowledge from Volumes of Data". In: *Communications of the ACM* 39.11 (1996), pp. 27–34. DOI: 10.1145/240455.240464.

[Gér22] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd. " O'Reilly Media, Inc.", 2022. ISBN: 9781492032649.

[Gim+22] N. L. Giménez et al. “Comparison of Two Microcontroller Boards for On-Device Model Training in a Keyword Spotting Task”. In: *2022 11th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2022, pp. 1–4. DOI: 10.1109/MECO55406.2022.9797171.

[Gol16] P. Goldsborough. “A Tour of TensorFlow”. In: *arXiv preprint arXiv:1610.01178* (2016). DOI: 10.48550/arXiv.1610.01178.

[Gu+18] J. Gu et al. “Recent Advances in Convolutional Neural Networks”. In: *Pattern recognition* 77 (2018), pp. 354–377. DOI: 10.1016/j.patcog.2017.10.013.

[HH21] Q. Huang and T. Hain. “Improving Audio Anomalies Recognition Using Temporal Convolutional Attention Networks”. In: *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, pp. 6473–6477. DOI: 10.1109/ICASSP39728.2021.9414611.

[Har23] S. Harris. *Inertial guidance: a brief history and overview*. Advanced Navigation. Jan. 3, 2023. URL: https://www.advancednavigation.com/tech-articles/inertial-guidance-a-brief-history-and-overview/ (visited on 12/09/2023).

[Idr15] I. Idris. *NumPy: Beginner’s Guide*. Packt Publishing Ltd, 2015. DOI: 9781785281969.

[Joh 1] J. Johnson. *Anomaly Detection with Machine Learning: An Introduction*. 12-12-2023. URL: https://www.bmc.com/blogs/machine-learning-anomaly-detection/.

[Kha21] R. Khandelwal. *A Basic Introduction to TensorFlow Lite - Towards Data Science*. https://towardsdatascience.com/a-basic-introduction-to-tensorfl 2021.

[Kum 1] S. Kumar. “5 Anomaly Detection Algorithms to Know”. In: *Built In* (8-11-2023). URL: https://builtin.com/machine-learning/anomaly-detection-algorithms.

[LG+22] N. Llisterri Giménez et al. “On-Device Training of Machine Learning Models on Microcontrollers with Federated Learning”. In: *Electronics* 11.4 (2022), p. 573. DOI: 10.3390/electronics11040573.

[Li+21] Z. Li et al. “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects”. In: *IEEE transactions on neural networks and learning systems* (2021). DOI: 10.1109/TNNLS.2021.3084827.

[Lsm] *LSM9DS1 Data Sheets*. Accessed Date 15.12.2023. 2015. URL: url:https://www.st.com/resource/en/datasheet/lsm9ds1.pdf.

[MB15] E. Mallon and P. Beddows. *Tutorial: How to Calibrate a Compass (and Accelerometer) with Arduino*. Accessed Date 15.12.2023. 2015. URL: https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-arduino/.

[Mis 1] A. Mishra. *Introduction to TinyML: What is it and Why does it Matter?* 11-12-2023. URL: https://circuitdigest.com/article/introduction-to-tinyml-what-is-it-and-why-does-it-matter.

[Mp3] *MP34DT06J – MEMS Audio Sensor Omnidirectional Digital Microphone*. Accessed Date 15.12.2023. 2021. URL: https://www.st.com/resource/en/datasheet/mp34dt06j.pdf.

[Nic 0] P. Nichani. "Outliers in Machine Learning - Analytics Vidhya - Medium". In: *Analytics Vidhya* (22-04-2020). URL: https://medium.com/analytics-vidhya/outliers-in-machine-learning-e830b2bd8660.

[Oor+16] A. v. d. Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *arXiv preprint arXiv:1609.03499* (2016). DOI: 10.48550/arXiv.1609.03499.

[Ous18] J. K. Ousterhout. *A Philosophy of Software Design*. Vol. 98. Yaknyam Press Palo Alto, CA, USA, 2018. ISBN: 9781732102200.

[PNW20] B. Pang, E. Nijkamp, and Y. N. Wu. "Deep Learning With TensorFlow: A Review". In: *Journal of Educational and Behavioral Statistics* 45.2 (2020), pp. 227–248. DOI: 10.3102/1076998619872761.

[Pan+15] V. Panayotov et al. "Librispeech: An ASR Corpus Based on Public Domain Audio Books". In: *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2015, pp. 5206–5210. DOI: 10.1109/ICASSP.2015.7178964.

[Raj19] A. Raj. *Arduino Nano 33 BLE Sense Review - What's New and How to Get Started?* Accessed Date 15.12.2023. 2019. URL: https://circuitdigest.com/microcontroller-projects/arduino-nano-33-ble-sense-board-review-and-getting-started-guide.

[Ray22] P. P. Ray. “A Review on TinyML: State-of-the-art and Prospects”. In: *Journal of King Saud University-Computer and Information Sciences* 34.4 (2022), pp. 1595–1623. DOI: 10.1016/j.jksuci.2021.11.019.

[Rib 1] J. Ribeiro. “What is TinyML, and Why does it Matter? | The AI Enthusiast”. In: *The AI Enthusiast* (22-12-2020). URL: https://medium.com/tech-cult-heartbeat/what-is-tinyml-and-why-does-it-matter-f5b164766876.

[SKP18] M. Sewak, M. R. Karim, and P. Pujari. *Practical Convolutional Neural Networks: Implement Advanced Deep Learning Models Using Python*. Packt Publishing Ltd, 2018. ISBN: 9781788392303.

[SM19] P. Singh and A. Manure. *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*. Apress, 2019. ISBN: 9781484255605.

[WS19] P. Warden and D. Situnayake. *TinyML – Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O’Reilly Media, Incorporated, 2019. ISBN: 9781492052043.

[Waq+21] D. M. Waqar et al. “Design of a Speech Anger Recognition System on Arduino Nano 33 BLE Sense”. In: *2021 IEEE 7th International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*. IEEE, 2021, pp. 64–69. DOI: 10.1109/ICSIMA50015.2021.9526323.

[War18] P. Warden. “Speech commands: A dataset for limited-vocabulary speech recognition”. In: *arXiv preprint arXiv:1804.03209* (2018). DOI: 10.48550/arXiv.1804.03209.

[Win23] E. Wings. *Knowledge Discovery in Databases Process*. Presentation for the Machine Learning course at the University of Applied Sciences Emden / Leer. 2023.

Index